# Common Vulnerabilities and Exposures: Analyzing the Development of Computer Security Threats

Andrew Kronser

May 18, 2020

Faculty of Science

University of Helsinki

**Supervisor(s)**

Prof. Nikolaj Tatti

**Examiner(s)**

Prof. Nikolaj Tatti

Dr. Razane Tajeddine

**Contact information**

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

| Tiedekunta — Fakultet — Faculty | |
|---|---|
| Faculty of Science | |

Tekijä — Författare — Author

Andrew Kronser

Työn nimi — Arbetets titel — Title

Common Vulnerabilities and Exposures: Analyzing the Development of Computer Security Threats

Ohjaajat — Handledare — Supervisors

Prof. Nikolaj Tatti

| Työn laji — Arbetets art — Level | Aika — Datum — Month and year | Sivumäärä — Sidoantal — Number of pages |
|---|---|---|
| | May 18, 2020 | 49 pages, 1 appendice pages |

Tiivistelmä — Referat — Abstract

Computer security professionals are at several disadvantages compared to the adversaries that seek to exploit computer systems. The Common Vulnerabilities and Exposures list was introduced in 1999 to make information more readily accessible in service of tool interoperability, risk sharing, and effective communication. The goal of this thesis is to leverage techniques in unsupervised learning and data mining in order to identify and visualize patterns in the development of threats over the lifespan of this list. We consider multi-dimensional clustering using $K$-medoids and hierarchical clustering. We also consider three methods for segmentation: segmentation optimized for an additive cost function using dynamic programming, monotonic segmentation of exponentially distributed data, and a method for multinomial change point detection that measures the divergence between static and dynamic parameter estimators.

We find that there is signal between the exploit types and products. Certain product types also experience more severe exploits on average. A monotonic segmentation of delay traces indicates that the number of identified threats proliferates over the supported lifespan of large products. Finally, the composition of threat types has notably diversified since 2018.

**ACM Computing Classification System (CCS)**
Computing methodologies → Machine learning → Learning paradigms → Unsupervised learning
Security and privacy → Software and application security

Avainsanat — Nyckelord — Keywords

security, segmentation, clustering

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

# Contents

# 1 Introduction

Computer security professionals are at several disadvantages compared to the adversaries that seek to exploit computer systems. In particular, the ability to share information between security professionals is more difficult due to the proprietary nature of many software products and the associated risk in making vulnerability data available for wider consumption. Attackers are seldom limited by these concerns.

With the stated goal of making information more readily accessible in service of tool interoperability, risk sharing, and effective communication, the Common Vulnerabilities and Exposures list was made public in 1999 [3]. Over the last 21 years, the CVE list has served as a common reference within the computer security community, and has expanded to catalog over 180,000 unique exploits.

CVE entries are monitored by package managers such as nodejs's `npm` [24] for automatic alerts and patching and are incorporated into security testing suites including Metasploit [22]. Having a common framework for identifying attacks has advanced the original mission behind the publication of the CVE list—making rapid communication about the development of security threats both possible and practical.

In addition, there is now a public trove of security data that covers numerous publicly known attacks over a period of 20 years. This dataset confers the added benefit of allowing researchers to discover patterns in the development of security threats. By understanding how the attack landscape is changing, vendors can harden their products against the most relevant threat types, vectors, and designs.

The goal of this thesis is to leverage techniques in unsupervised learning and data mining in order to identify and visualize patterns in the development of threats over the lifespan of the CVE list. First, we will explore how data is formatted in the CVE list and associated sources and cover any necessary preprocessing.

Then we will compare two clustering techniques: $K$-Medoids and Hierarchical Clustering to examine the relationship between categorical features in the CVE data. In particular, we focus on whether products can be organized according to shared exploit characteristics. We also use kernelized principal component analysis to represent data in two dimensions.

In the following chapter, we consider segmentation and change point detection, variants on

a one-dimensional clustering problem. We outline an approach for finding optimal $K$ segmentations using dynamic programming, an approach for finding monotonic segmentations in exponentially distributed data, and a change point detection method for multinomially distributed data. These methods are used to analyze severity, delay between exploits, and categorical data (exploit types, attack patterns, severity factors) respectively.

Together, these experiments help to identify how the CVE list has changed over the course of its history. These changes illustrate both static and dynamic aspects of the current security landscape.

This thesis is organized as follows. In Chapter 2, we introduce and summarize the CVE data. In Chapter 3, we cover clustering to examine the relationship between categorical features using $K$-medoids and hierarchical clustering. In Chapter 4, we consider three segmentation methods: a dynamic programming solution, a monotonic segmentation solution, and Multinomial Change Detection Method, an online change point detection algorithm for categorical data. We identify related work in Chapter 5 and conclude with a discussion in Chapter 6.

# 2 CVE data

Common Vulnerabilities and Exposures (CVE) is a list of computer security threats provided by the U.S. Department of Homeland Security and maintained by the MITRE corporation.

Per MITRE's terminology, CVE distinguishes between vulnerabilities where

> A "vulnerability" is a weakness in the computational logic (e.g., code) found in software and some hardware components (e.g., firmware) that, when exploited, results in a negative impact to confidentiality, integrity, OR availability. Mitigation of the vulnerabilities in this context typically involves coding changes, but could also include specification changes or even specification deprecations (e.g., removal of affected protocols or functionality in their entirety).

and exposures where

> An "exposure" is a system configuration issue or a mistake in software that allows access to information or capabilities that can be used by a hacker as a stepping-stone into a system or network.
>
> CVE considers a configuration issue or a mistake an exposure if it does not directly allow compromise but could be an important component of a successful attack, and is a violation of a reasonable security policy.

though both vulnerabilities and exposures are present in the CVE list [20].

CVE data entries are distributed as part of the National Vulnerability Database (NVD) where they are augmented to reflect the severity of their impact and any available fixes. For the purposes of this paper, CVE data will be used to refer to both the CVE entry itself and any associated data from the NVD or other authoritative sources.

CVE data was obtained from the hosted JSON files at The Computer Incident Response Center Luxembourg (CIRCL) through their cve-search project (http://cve-search.org/dataset/). This data is updated daily and the figures in this thesis represent the data dump from April 13, 2020.

```
{
    "id":"CVE-2010-3333"
    "Modified": "2018-10-12T21:58:00",
    "Published": "2010-11-10T03:00:00",
    "access": {
        "authentication": "NONE",
        "complexity": "MEDIUM",
        "vector": "NETWORK"
    },
    "cvss": 9.3,
    "cvss-time": "2018-10-12T21:58:00",
    "cvss-vector": "AV:N/AC:M/Au:N/C:C/I:C/A:C",
    "cwe": "CWE-119",
    "id": "CVE-2010-3333",
    "impact": {
        "availability": "COMPLETE",
        "confidentiality": "COMPLETE",
        "integrity": "COMPLETE"
    },
    "summary": "Stack-based buffer overflow in Microsoft Office XP SP3,
    Office 2003 SP3, Office 2007 SP2, Office 2010, Office 2004 and 2008
    for Mac, Office for Mac 2011, and Open XML File Format Converter for
    Mac allows remote attackers to execute arbitrary code via crafted RTF
    data, aka \"RTF Stack Buffer Overflow Vulnerability.\"",
}
```


**Figure 2.1:** A sample JSON entry for CVE-2010-3333

## 2.1 Features

Next we are going to describe the features selected for this thesis (outlined in Table 2.1). We group features in four categories: edit time log, exploit class, text, and severity factors.

### 2.1.1 CVE IDs

Though not a feature proper, entries are indexed by their CVE ID. The CVE ID is a unique identifier assigned to a particular vulnerability or exposure by a CVE Numbering Authority (CNA). These organizations represent governments, industry response teams, and affected vendors. At the time of writing, there are 116 such CNAs representing 22

**Table 2.1:** Selected CVE features split into four groups

| Group | Attribute | Type | Description |
|---|---|---|---|
| Edit time log | publication date | datetime | time the entry was published |
| | modification date | datetime | time the entry was last modified |
| Exploit class | CWE name | text | Common Weakness Enumeration classification |
| | CWE code | int | unique id for the associated CWE classification |
| Text | summary | text | a short description of the exploit |
| Severity factors | CVSS | float | a severity score (0–10) |
| | access authentication | categorical | {NONE, SINGLE, MULTIPLE} |
| | access complexity | categorical | {LOW, MEDIUM, HIGH} |
| | access vector | categorical | {LOCAL, NETWORK, ADJ. NETWORK} |
| | availability impact | categorical | {NONE, PARTIAL, COMPLETE} |
| | confidentiality impact | categorical | {NONE, PARTIAL, COMPLETE} |
| | integrity impact | categorical | {NONE, PARTIAL, COMPLETE} |

countries.

CVE IDs are currently issued with the following format: the CVE prefix followed by the year and the sequence number. Sequence numbers are at least 4 digits long, but can be an arbitrary length. The only sequence numbers prefixed with leading zeroes are 1–999. Sequence numbers can be reserved for later use, but are otherwise issued in order [21]. For example CVE-2010-3333 was issued before the format change (in 2010) and has been assigned sequence number 3333.

### 2.1.2 Edit time log

The number of known threats has continued to grow since the system's release in September 1999. Understanding how the nature and scope of threats has changed over time requires an ordinal feature. For the purposes of this thesis, CVE entries are ordered by publication date. Entries may be modified after their publication, and for this purpose a modification date is also published along with the associated changelog. The experiments in this paper, however, do not explicitly account for modification.

Figure 2.2 illustrates the distribution of threats and how its growth has changed over time. The proliferation of new threats has leveled over the course of the last decade. Due to

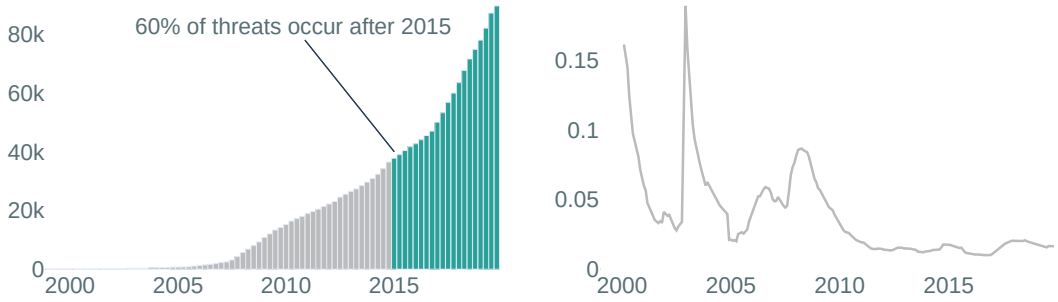



**Figure 2.2:** Threat Proliferation: The number of known threats (left) continues to grow, but growth (right), measured as a percent change on a 12-month rolling average, has leveled.

sparsity of data before 1999, all entries made before the list's public release have been dropped from consideration for the experiments in this thesis.

### 2.1.3 Exploit class

Each CVE entry is associated with a weakness classification according to the Common Weakness Enumeration (CWE) which is also published and maintained by the MITRE Corporation. The goal of this list is to categorize how threats affect products to help identify and fix related exploits. The CWE name is most often a succinct description of how the exposure or vulnerability affects the system. For example, CVE-2010-3333 is associated with CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer.

CWE names can also reference hierarchical or archetypal descriptions of threats. Several extant taxonomies (e.g. Seven Pernicious Kingdoms) of CWEs have their own CWE entries to represent these hierarchies. A CVE threat can also be assigned a higher-order CWE classification if the lower-order classifications fail to describe its behavior.

There are three placeholders used to indicate a null CWE in the data. These are `Unknown`, `NVD-CWE-Other`, and `NVD-CWE-noinfo`. Since CWE codes are used in most of the experiments in Chapters 2 and 3, null valued CWE codes are dropped from consideration.

Figure 2.3 shows the trends among the top-ten (by frequency in the entire CVE list at the time of writing) CWEs. The trends shown indicate that the relative prominence of CWE

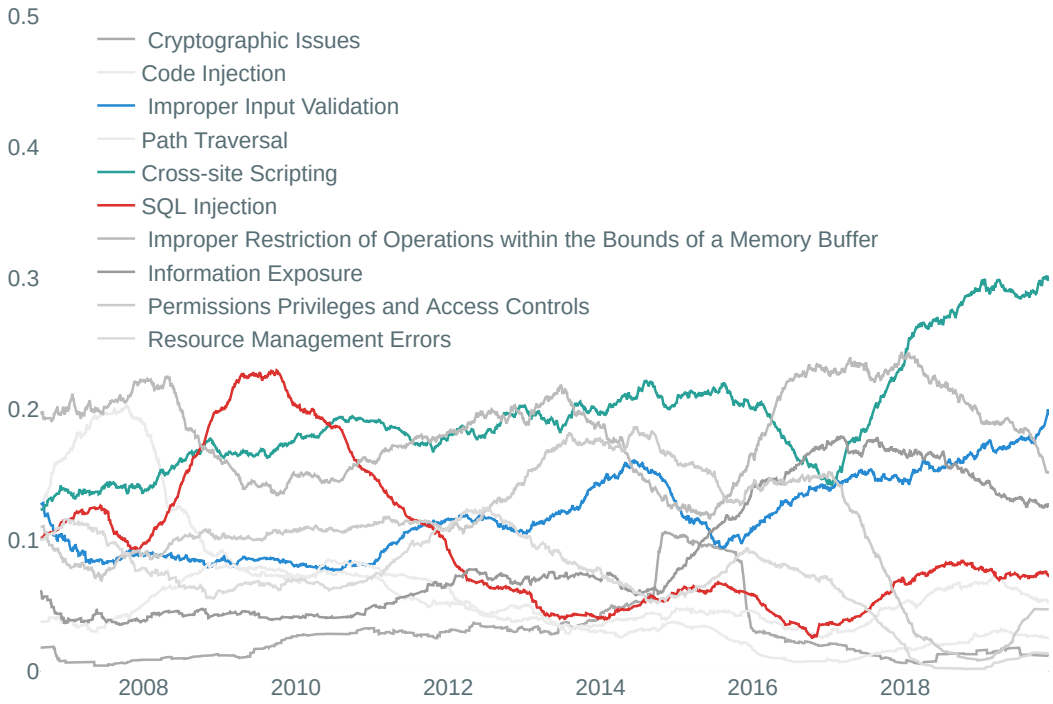



**Figure 2.3:** CWE Trends: These threats are the 10 most common, but their relative prominence is shifting, SQL injection is becoming less common while cross-site scripting and improper input validation are on the rise. Values are shown as a 365-entry rolling average of relative frequencies.

codes is dynamic over time, with different attacks falling in and out of fashion.

### 2.1.4 Description

Each CVE entry is given a text description. This description, 40 words on average, is provided by the CNA at the time of assignment. It may contain a list of affected products, a short description of the attack, affected files, and a overview of the attack's impact though the content varies significantly.

There are three marks that can be added to a CVE description. The first, `RESERVED`, indicates that a CNA is reserving a CVE ID for later use. Usually this label is only held while the details of an entry are being populated. The second, `REJECT`, indicates a CVE that has been removed from the CVE list, usually for administrative reasons (e.g. a duplicate entry or an inaccurate entry removed by the issuing CNA). The third, `DISPUTED`,

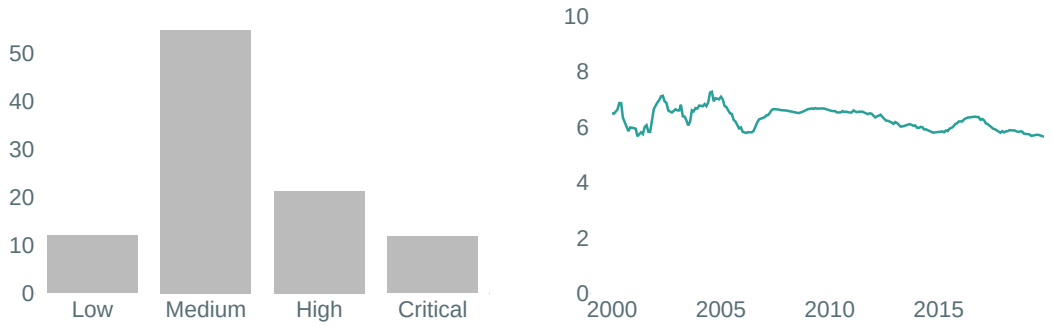



**Figure 2.4:** Threat Severity Distribution: CVSS scores reflect a threat's severity. Over 50 percent of scores (left) fall in FIRST's Medium threat category with a thicker tail toward the higher end of the spectrum. The average severity (right), shown as a 12-month rolling average, has remained largely constant.

indicates that parties disagree as to whether or not an entry is a vulnerability. For the experiments in this paper, `RESERVED` and `REJECT` entries are dropped from the dataset, but `DISPUTED` entries remain.

### 2.1.5 Severity

The Common Vulnerability Scoring System (CVSS) is an open framework for describing the characteristics and severity of computer security exploits developed and maintained by the Forum of Incident Response and Security Teams (FIRST). Scores range from 0–10 and can also be assessed using a qualitative, categorical scale: None (0.0), Low (0.1–3.9), Medium (4.0–6.9), High (7.0–8.9), Critical (9.0–10.0). Though this scale was introduced in the version 3 specification, it applies retroactively [7]. The distribution of these qualitative labels and the trend of the quantitative score are shown in Figure 2.4.

Scores are calculated by combining several categorical metrics. Each metric is assigned according to a scoring rubric. The complete list of factors (for CVSS version 2) can be seen in Figure 2.6, however, we only consider the CVSSv2 base score and the six base (non-optional) metrics: Access Vector, Access Complexity, Authentication, Confidentiality Impact, Integrity Impact, and Availability Impact. Trends in these metrics are presented in Figure 2.5 as annualized averages.

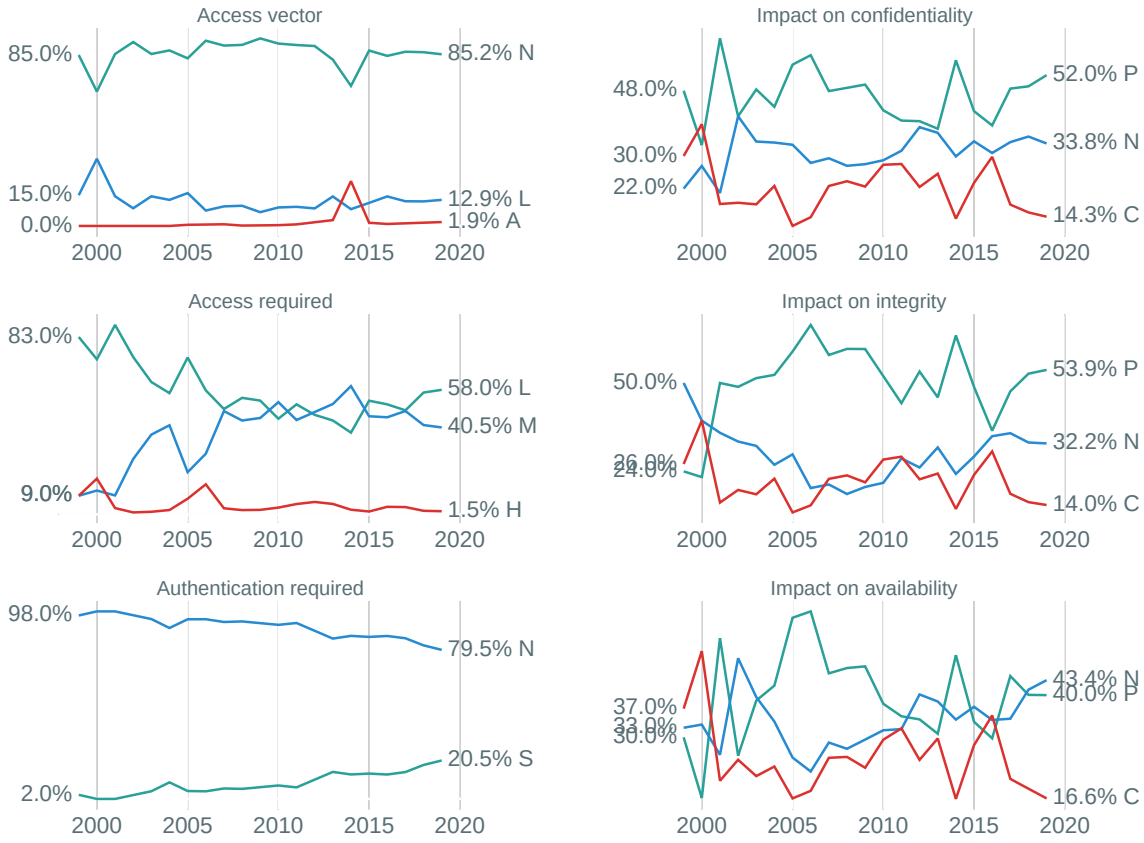



**Figure 2.5:** Base metric trends: all three impacts and the access vector are volatile and don't show a clear long term trend; the access and complexity required, however, is steadily increasing.

No data present has a null CVSS score, though data may be missing any of the six base metrics. Rows with null values are retained except for calculations and transformations that explicitly require non-null metrics.

We provide the formulas for computing CVSSv2 scores in Appendix A. For example, CVE-2010-3333 was rated Severe (with a score of 9.3) since it allowed arbitrary code execution. More specifically, it completely compromised all three of the impact metrics while requiring only moderately complex network access with no required authentication.

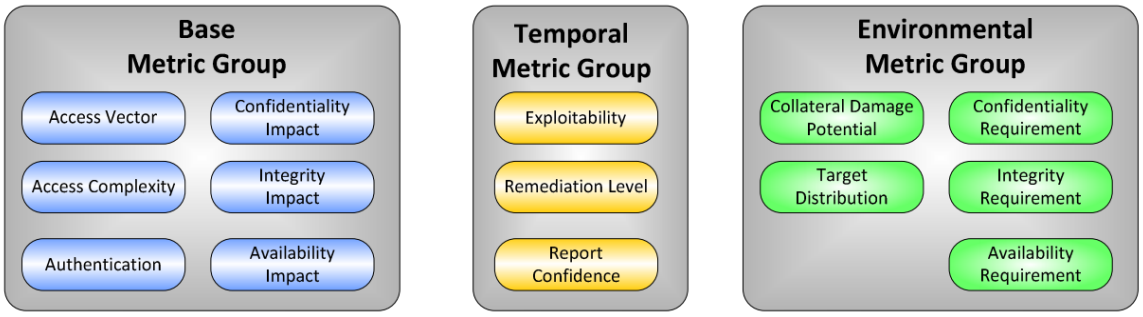



**Figure 2.6:** Components of CVSS (from FIRST [6] Figure 1)

### 2.1.6 Exploitability metrics

Exploitability metrics (access vector, access complexity, and authentication) characterize how attacks affect their targets. They comprise 40% of the final CVSS Base Score.

#### Access Vector

**Table 2.2:** The rubric for Access Vector (from FIRST [6] Table 1)

| Name | Initial | Description |
|---|---|---|
| Local | L | A vulnerability exploitable with only local access requires the attacker to have either physical access to the vulnerable system or a local (shell) account. Examples of locally exploitable vulnerabilities are peripheral attacks such as Firewire/USB DMA attacks, and local privilege escalations (e.g., sudo). |
| Adjacent Network | A | A vulnerability exploitable with adjacent network access requires the attacker to have access to either the broadcast or collision domain of the vulnerable software. Examples of local networks include local IP subnet, Bluetooth, IEEE 802.11, and local Ethernet segment. |
| Network | N | A vulnerability exploitable with network access means the vulnerable software is bound to the network stack and the attacker does not require local network access or local access. Such a vulnerability is often termed "remotely exploitable". An example of a network attack is an RPC buffer overflow. |

An access vector is the avenue by which an attacker exploits a system. There are three access vectors in CVSSv2: Local (L), Adjacent Network (A), and Network (N). The rubric for assigning an attack vector is shown in Table 2.2.

### Access Complexity

**Table 2.3:** The rubric for Access Complexity (from FIRST [6] Table 2 with some examples omitted)

| Label | Initial | Description |
|---|---|---|
| High | H | Specialized access conditions exist. For example:<br>– In most configurations, the attacking party must already have elevated privileges or spoof additional systems in addition to the attacking system.<br>– The attack depends on social engineering methods that would be easily detected by knowledgeable people.<br>– The vulnerable configuration is seen very rarely in practice. |
| Medium | M | The access conditions are somewhat specialized; the following are examples:<br>– The attacking party is limited to a group of systems or users at some level of authorization, possibly untrusted.<br>– The affected configuration is non-default, and is not commonly configured.<br>– The attack requires a small amount of social engineering that might occasionally fool cautious users. |
| Low | L | Specialized access conditions or extenuating circumstances do not exist. The following are examples:<br>– The affected product typically requires access to a wide range of systems and users, possibly anonymous and untrusted.<br>– The affected configuration is default or ubiquitous. |

Access complexity is a summary of how complex an attack must be to exploit a system. There are three levels of access complexity in CVSSv2: High (H), Medium (M), and Low (L). The rubric for assigning an access complexity is shown in Table 2.3

### Authentication

Authentication records how many times an attacker must obtain authentication for the exploit to succeed. Possible values are Multiple (M), Single (S), and None (N). The rubric is shown in Table 2.4

**Table 2.4:** The rubric for Authentication (from FIRST [6] Table 3 with an example omitted)

| Label | Initial | Description |
|---|---|---|
| Multiple | M | Exploiting the vulnerability requires that the attacker authenticate two or more times, even if the same credentials are used each time. |
| Single | S | The vulnerability requires an attacker to be logged into the system (such as at a command line or via a desktop session or web interface). |
| None | N | Authentication is not required to exploit the vulnerability. |

### 2.1.7 Impact metrics

Impact metrics measure the resultant impact of an attack on any affected targets. Impacts are all scored with the same values: None (N), Partial (P), and Complete (C). Impact assessments comprise 60% of the final base score.

#### Confidentiality Impact

**Table 2.5:** The rubric for Confidentiality Impact (from FIRST [6] Table 4)

| Label | Initial | Description |
|---|---|---|
| None | N | There is no impact to the confidentiality of the system. |
| Partial | P | There is considerable informational disclosure. Access to some system files is possible, but the attacker does not have control over what is obtained, or the scope of the loss is constrained. An example is a vulnerability that divulges only certain tables in a database. |
| Complete | C | There is total information disclosure, resulting in all system files being revealed. The attacker is able to read all of the system's data (memory, files, etc.) |

Confidentiality is the protection of information from unauthorized disclosure. Confidentiality impact measures the degree to which an attack resulted in information disclosure. The rubric for scoring confidentiality impact is shown in Table 2.5.

#### Integrity Impact

Integrity is a measure of a system's ability to protect itself. If integrity is compromised, the affected product can no longer make guarantees about how it performs or is configured. The rubric for scoring integrity impact is shown in Table 2.6.

**Table 2.6:** The rubric for Integrity Impact (from FIRST [6] Table 5)

| Label | Initial | Description |
|---|---|---|
| None | N | There is no impact to the integrity of the system. |
| Partial | P | Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited. For example, system or application files may be overwritten or modified, but either the attacker has no control over which files are affected or the attacker can modify files within only a limited context or scope. |
| Complete | C | There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised. The attacker is able to modify any files on the target system. |

### Availability Impact

Availability measures whether resources can be accessed by clients. As a system's availability suffers, clients can no longer access affected resources. The rubric for scoring availability impact is shown in Table 2.7.

**Table 2.7:** The rubric for Availability Impact (from FIRST [6] Table 6)

| Label | Initial | Description |
|---|---|---|
| None | N | There is no impact to the availability of the system. |
| Partial | P | There is reduced performance or interruptions in resource availability. An example is a network-based flood attack that permits a limited number of successful connections to an Internet service. |
| Complete | C | There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable. |

## 2.2 Associated data

In addition to the above data, each CVE entry can be associated with multiple attack patterns and affected products.

### 2.2.1 Attack patterns

Released in 2007, the Common Attack Pattern Enumeration and Classification (CAPEC) list is another part of the Mitre corporation and their partners' efforts to chronicle computer security threats. An attack pattern, inspired by the concept of design patterns from software engineering, is a description of methods used by attackers across multiple security exploits.

Common examples of attack patterns include, XSS in HTTP query strings and rainbow table password cracking. Unlike the CWE, which focuses on the effects of security threats, CAPEC describes common ways that exploits are designed with the intent of sharing ways to mitigate these same patterns.

### 2.2.2 Affected products

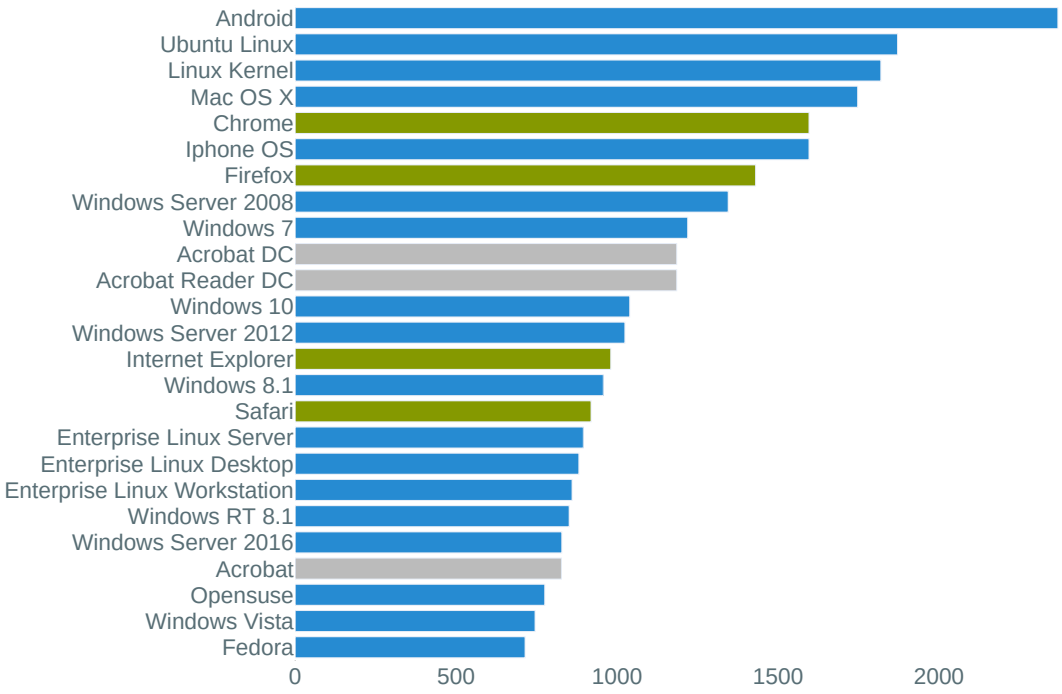



**Figure 2.7:** Top affected products: the number of unique CVE entries per product is shown for the top 25 affected products, most of which are operating systems (blue) or web browsers (green)

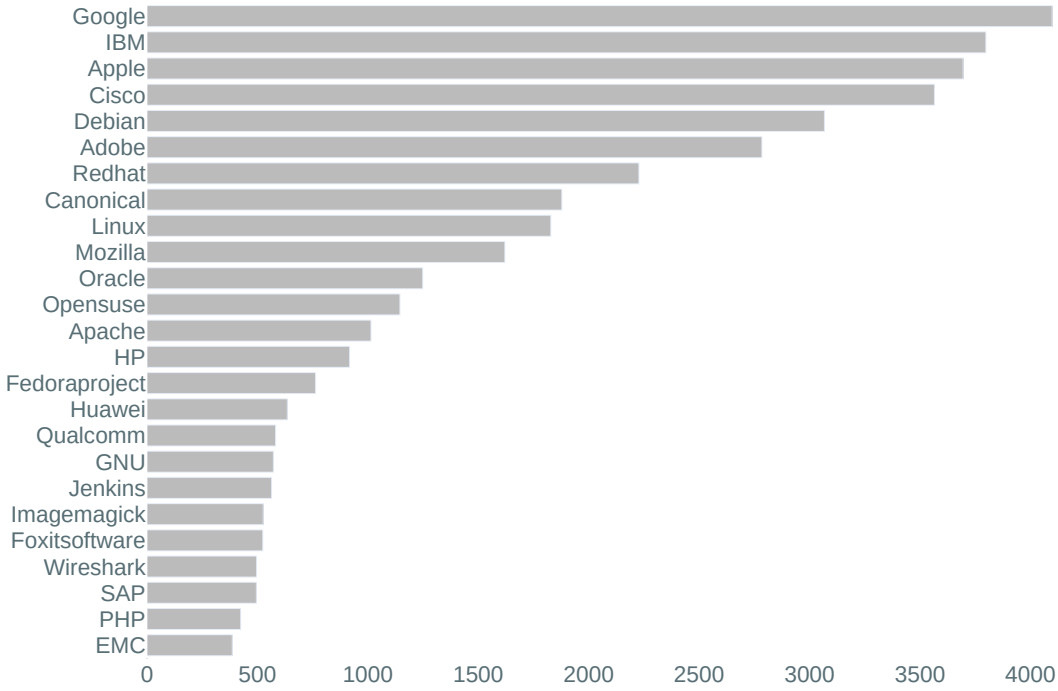



**Figure 2.8:** Top affected vendors: the number of unique CVE entries per vendor is shown for the top 25 affected vendors.

The products affected by a given CVE entry are listed according to the Common Platform Enumeration (CPE) which is maintained by the National Institute of Standards and Technology (NIST) at the time of writing. The CPE was formerly maintained by the Mitre Corporation.

The CVE data provided by CIRCL lists products in CPE 2.3 formatted string bindings. These bindings are part of the naming layer of CPE [23]. Though all of the names present are software names, CPE also extends to hardware naming. Each formatted string is a colon separated string,

```
cpe:2.3: part : vendor : product : version : update : edition :
         language : sw_edition : target_sw : target_hw : other
```

Not all fields need to be specified and a wildcard `*` can be used. For the purposes of this paper, only vendor, product, and version information will be retained. The top 25 products by unique CVE count are shown in Figure 2.7; the top 25 vendors are shown

in Figure 2.8. CVE associated data is many-to-many in nature, a single CVE can affect

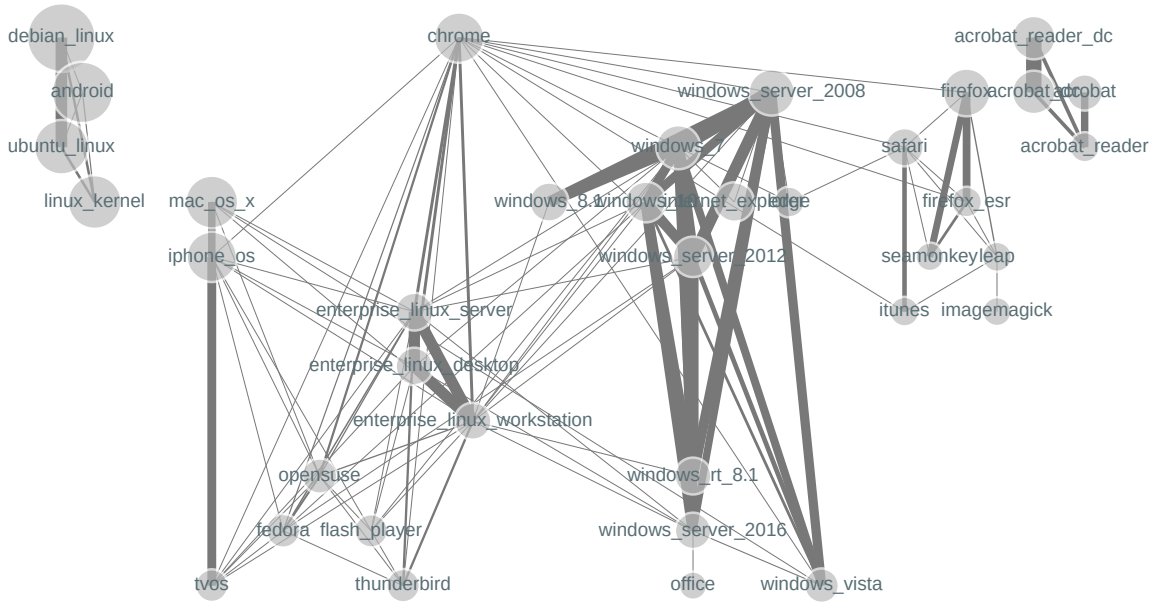



**Figure 2.9:** Shared CVEs: products with more than 500 unqiue CVEs are represented as graph nodes (scaled to reflect the number of CVEs that affect a given product). Edges represent the number of CVE entries a pair of products have in common.

multiple products and a single product may be affected by multiple CVE entries. As we consider clusterings across products, note that some of the aggregations will include the same CVE entry multiple times for different products. Figure 2.9 represents which products have CVEs in common as a weighted graph.

# 3 Clustering

Clustering is a family of unsupervised methods where data are divided into self-similar groups called clusters. In this section, we will examine two clustering methods: $K$-medoids and hierarchical clustering.

We will apply clustering to analyze the relationship between different categorical features in order to better understand how they influence one another. In particular, we will examine how products relate to exploit types and how products relate to attack patterns. These relationships will be examined in both directions, for example, clustering both products by exploit type and exploit types by product.

First, in Sections 3.1–3.2, we will discuss how we represent categorical data and project them into 2-dimensional space. Then, we will discuss clustering according to $K$-medoids in Section 3.3 and hierarchical clustering in Section 3.4. At the end of the chapter, in Section 3.5, we will compare these two clustering methods.

## 3.1 Representing categorical data

Categorical data are data that are divided into groups called categories. A categorical feature $F_i$ is a sequence of labels $F_i = x_i^{(1)}, \ldots, x_i^{(N)}$. The label set $L_i = \{x_i^{(j)} \in F_i\}$ is the set of unique labels that occur in the sequence.

To examine the relationship between two categorical features $F_A$ and $F_B$ indexed by a common index set $I = \{1, \ldots, N\}$, we use the normalized co-occurrence matrix, $Y_A^B$. $Y_A^B$, is a $|L_\mathrm{A}| \times |L_\mathrm{B}|$ matrix whose entries

$$Y_a^b = \frac{|\{i \in I; x_A^{(i)} = a, x_B^{(i)} = b\}|}{|\{i \in I; x_A^{(i)} = a\}|}$$

represent the relative frequencies of each label in $L_\mathrm{B}$ for each label in $L_\mathrm{A}$. Note that each row sums to 1. For example, $Y_{\text{product}}^{\text{CWE}}$ considers the relative frequency of CWE codes aggregated according to products.

**Table 3.1:** Kernels considered in this paper and included in `sklearn's` implementation of Kernel PCA. Additional parameters: $\gamma$ is a scale parameter, $c_0$ is a constant additive or intercept, $d$ is the dimensionality of $x$.

| Kernel name | $K(x, x')$ |
|---|---|
| Linear | $x \cdot x'$ |
| Polynomial | $(\gamma x \cdot x' + c_0)^d$ |
| Radial Basis Function | $\exp(-\gamma \|\|x - x'\|\|^2)$ |
| Cosine | $\frac{x \cdot x'}{\|\|x\|\|\|\|x'\|\|}$ |

## 3.2 Kernel PCA

Principal component analysis (PCA) is used in this thesis as a dimensionality reduction technique to project clusterings into 2-dimensional space. PCA finds a projection in such a way as to maximize the variance retained from the original data. To perform the transformation, data are first zero-centered, and then an eigen-decomposition of the covariance matrix yields eigenvectors and eigenvalues. The eigenvectors that correspond to the highest valued eigenvalues are the principal components [32]. To achieve dimensionality reduction, data points are projected onto these principal components.

In this thesis, we consider a kernalized version of PCA. Kernel methods allow us to represent data in an alternate space, $Z$. Let $\phi : X \to Z$ be a function that maps input from the original space, $X$ to an alternate space, $Z$. The goal of Kernel PCA is to map the data according to $\phi$ and then perform PCA. To accomplish this, we introduce a kernel function. Let the kernel function $K(x, x')$ be equivalent to the inner product $\phi(x) \cdot \phi(x')$. This inner product can be used in the formulation of many popular machine learning algorithms without ever needing to project data into the new space. This is referred to as the kernel trick. Thus, without needing to represent $\phi(x)$ or $\phi(x')$ explicitly, we represent similarities between $\phi(x)$ and $\phi(x')$ [11].

The kernels considered in this thesis are outlined in Table 3.1.

Kernel PCA differs from linear PCA in that first a kernel function $K(x, x')$ is selected (which maps to the inner product space of some feature space $Z$ identified by a feature map $\phi(x)$). Rather than using data in the feature space (which may be infinitely large even for finite data), instead, we construct a kernel matrix $K$ whose entries, $K_{i,k}$, correspond

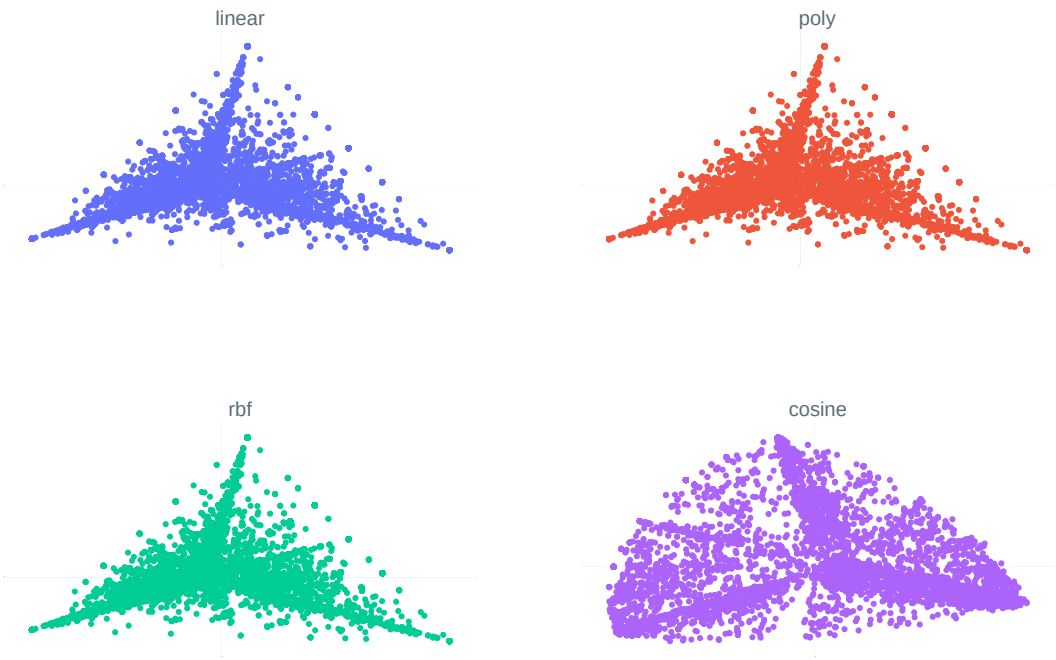



**Figure 3.1:** Kernel comparison: each entry is a product represented by exploit types projected according to four different kernels.

to $K(x_i, x_k)\ \forall\ x_i, x_k \in \mathcal{D} \times \mathcal{D}$.

Then solving the eigenvector equation $N\lambda\alpha = K\alpha$ yields the principal components $\alpha$ and eigenvalues $\lambda$ in the kernel space [18]. In this chapter, we consider different kernels for projecting categorical features into 2-dimensions for visualizing later clusterings. We will use the cosine kernel for this purpose since it disperses examples more than the other kernels.

## 3.3 $K$-Medoids

$K$-Medoids is a clustering problem in the same family as the $K$-means problem [27]. $K$-Medoids aims to partition the data into $K$ clusters, $C = \{C_1, \ldots, C_K\}$ with corresponding cluster centers $\mu_i$ called medoids, chosen to minimize the total cost

$$\sum_{i=1}^{K} \sum_{x \in C_i} d(x, \mu_i),$$

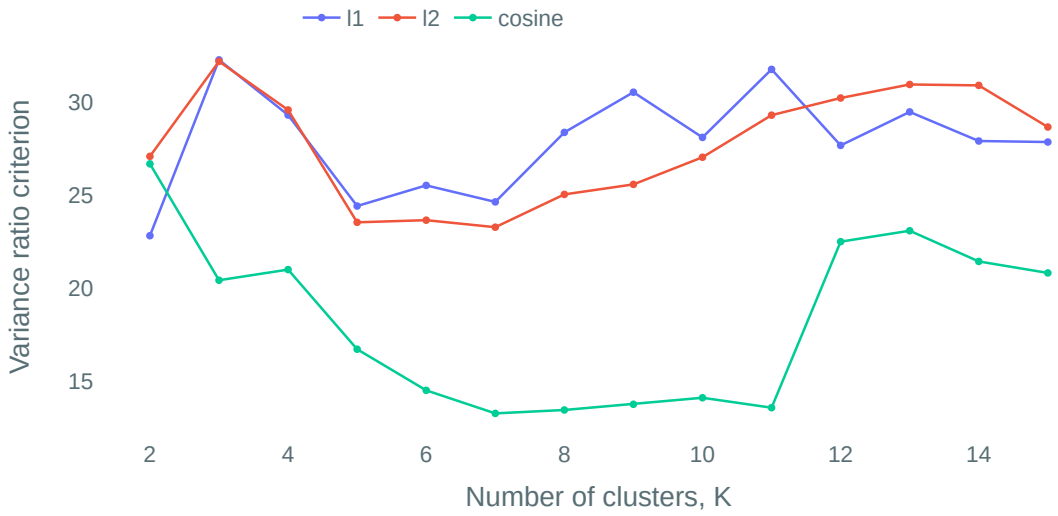



**Figure 3.2:** Variance ratio criterion: the ratio of within-cluster variance to between cluster variance for our three distance measures

given some distance function $d$. Since medoids are defined according to a distance function, we can use cosine dissimilarity since it corresponds to the kernel chosen in the last section.

The algorithm commonly used to approximate $K$-medoids is the partition around medoids or PAM algorithm [27]. The algorithm starts by selecting $K$ data points to be the initial medoids. Similar to Lloyd's algorithm for $K$-means, in the next step, points are assigned to the closest medoid. Then the algorithm greedily selects swapping a medoid with a non-medoid point in the dataset such that the overall cost is reduced. If no such point exists, the algorithm is considered converged.

### 3.3.1 Selecting $K$ and $d$

The distance function $d : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$ characterizes how similar two data points are to one another. Common distance functions (defined in Table 3.2) are the $L_2$ metric (distance between points in Cartesian space) and the $L_1$ metric (the sum of the absolute differences between points). Non-metric distance functions such as cosine dissimilarity (1 minus the cosine of the angle between two vectors) can also be used.

We select both $d$ and $K$ according to the variance ratio criterion. This statistic measures the ratio of within-cluster variance to between-cluster variance [2]. The first local maxi-

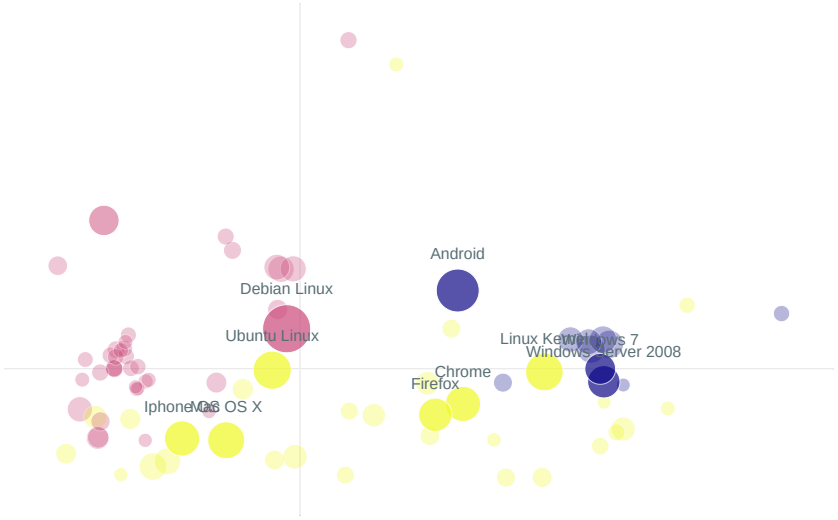



**Figure 3.3:** A 3-medoids clustering of products according to their exploit types, $Y^{\text{CWE}}_{\text{product}}$ with L1 distance. Size is scaled relative to the number of CVE entries for each product and the top 10 products are darkened and labelled.

**Table 3.2:** Common distance functions

| Function name | $d(a, b)$ |
|---|---|
| $L_1$ metric | $\sum_{i=1}^{N} \lvert a_i - b_i \rvert$ |
| $L_2$ metric | $\sum_{i=1}^{N} \sqrt{(a_i - b_i)^2}$ |
| Cosine dissimilarity | 1 - $\dfrac{\sum_{i=1}^{N} a_i, b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}}$ |

mum is used to select $K$ and $d$. Here, for both clusterings $K = 3$ and $d$ is the $L_1$ metric. The product by exploit type clustering variance ratio criterion is shown for varying values of $K$ and $d$ in Figure 3.2.

### 3.3.2 Exploit type and products

Figure 3.3 shows a 3-medoids clustering of products according to the relative frequencies of exploit types. In this clustering, some product lines are clustered and plotted close together (e.g. Windows operating systems) and other very similar products are split between clusters (Debian and Ubuntu for example). Overall, the data is not well separated.

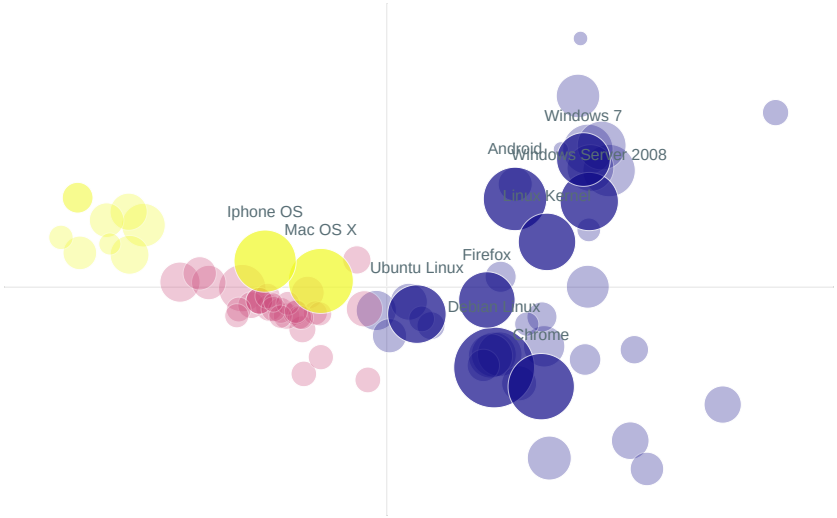



**Figure 3.4:** A 3-medoids clustering of products according to attack patterns, $Y^{\text{CAPEC}}_{\text{product}}$ with L1 distance. Size is scaled relative to the number of CVE entries for each product and the top 10 products are darkened and labelled.

**Table 3.3:** Common linkage criteria

| Linkage criterion | $\mathcal{L}(A, B, d)$ |
|---|---|
| Single linkage | $\min_{a \in A, b \in B} d(a, b)$ |
| Complete linkage | $\max_{a \in A, b \in B} d(a, b)$ |
| Average linkage | $\frac{1}{N} \sum_{a \in A, b \in B} d(a, b)$ |

### 3.3.3 Attack patterns and products

Figure 3.4 shows products clustered according to the relative frequencies of attack patterns. Unlike clustering products by exploit type, most large products are now grouped into a single cluster. Products from the same vendor and products in the same product line are mostly clustered together.

## 3.4 Hierarchical clustering

Hierarchical Clustering is a clustering method where data points are grouped into a cluster hierarchy [14]. For the bottom-up (agglomerative) variant, initially each observation is considered a singleton cluster. At each iteration, the two most similar clusters are merged

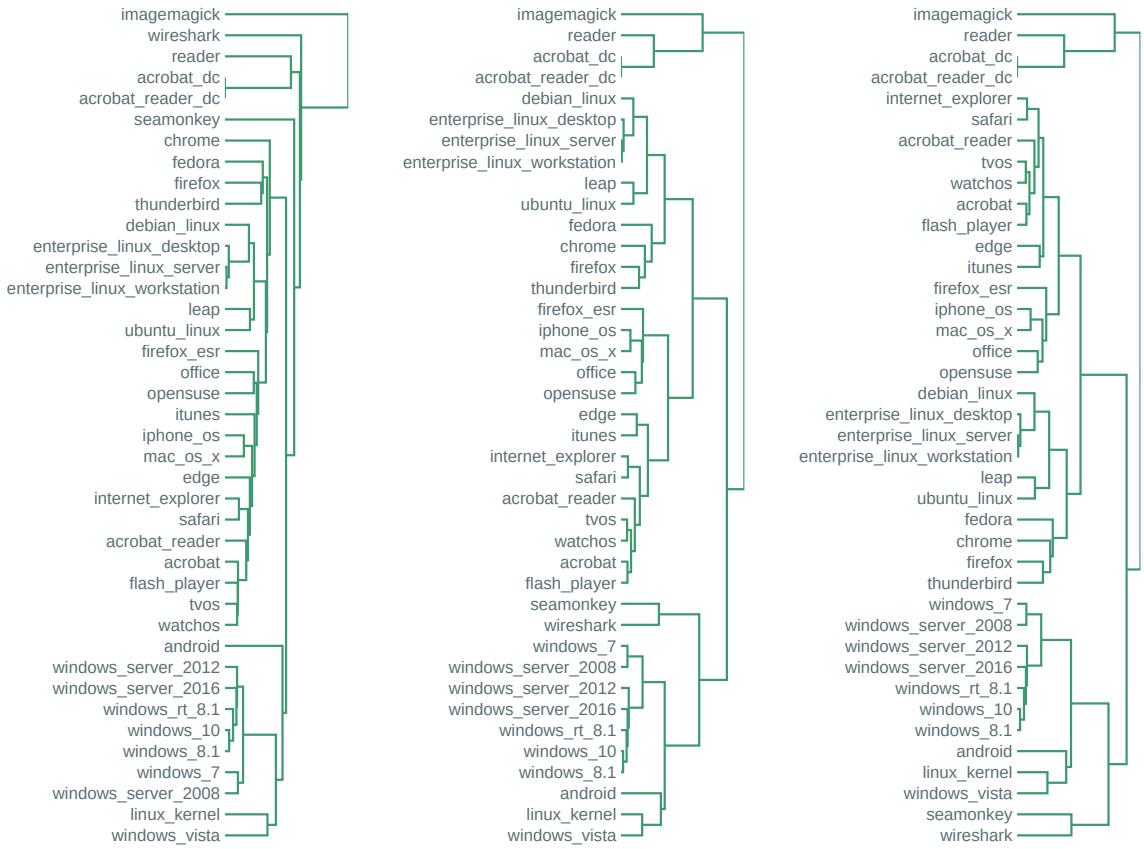



**Figure 3.5:** Hierarchical clustering for the top 40 products according to exploit type, $Y^{\text{CWE}}_{\text{product}}$ with cosine dissimilarity for (a) complete, (b) single, and (c) average linkage.

until the desired number of clusters remain. The primary output of this process is a dendrogram, a tree-representation of the successive cluster merge operations. Similar entries occur next to one another and the height of each merge indicates the similarity between the clusters being merged. A clustering is determined by a distance function and a linkage criterion.

Distance functions, like in $K$-Medoids, measure the distance between two datapoints. During the process, we also need to compute the distance between clusters. In order to do that, we use a linkage criterion $\mathcal{L}$ to extend the notion of distance to clusters. Popular choices, defined in Table 3.3, include single linkage, complete linkage, and average linkage.

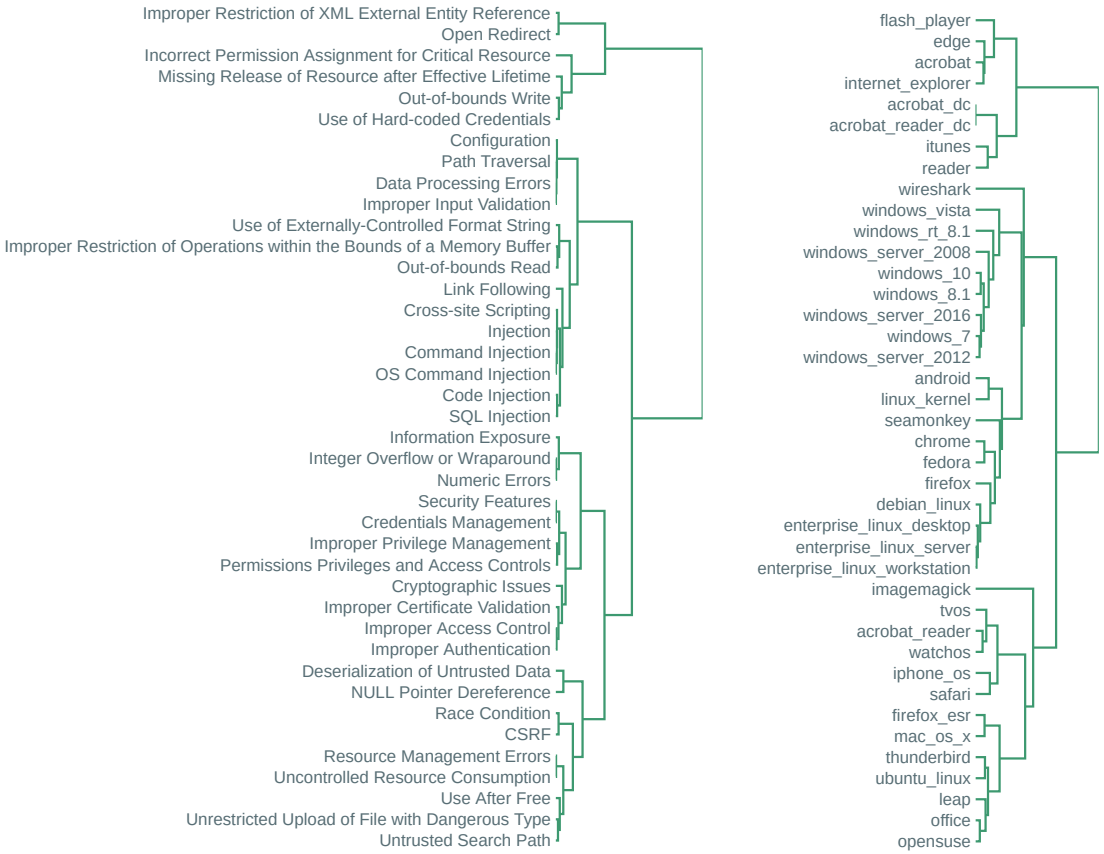
In single linkage clustering, the similarity of two clusters is equivalent to the distance between the closest two points. In complete linkage clustering, similarity is instead equivalent


**Figure 3.6:** Hierarchical clustering for the top 40 exploit types according to product $Y_{\text{CWE}}^{\text{product}}$ with cosine dissimilarity and average linkage

to the distance between the furthest two points.

### 3.4.1 Exploit type and products

Figure 3.5 compares the three linkage criteria in Table 3.3 for clustering products according to the relative frequency of exploit types. Both single and average linkage group the top 40 products into three high-level groups. Windows operating systems along with Seamonkey (an internet suite provided by Mozilla), Wireshark (a network protocol analyzer), the Linux kernel, and Android form one group. In the second group, fall most of the web browsers (IE, Firefox, Chrome, Safari, Edge) along with Linux distributions (OpenSUSE, Debian, Ubuntu) and Apple's operating systems. The third and smallest group contains the document cloud versions of Adobe's reader and Imagemagick.




(a) (b)

**Figure 3.7:** Hierarchical clustering for the (a) top 40 CAPECS $Y^{\text{product}}_{\text{CAPEC}}$ according to product and (b) top 40 products according to CAPECs $Y^{\text{CAPEC}}_{\text{product}}$ with cosine dissimilarity and average linkage

Figure 3.6 shows exploit types clustered according to product frequencies. Some similar exploits (e.g. injection exploits, integer overflow/numeric errors) occur within close cluster hierarchy whereas other seemingly unrelated attacks (cross-site request forgery and race condition) also occur in close proximity.

### 3.4.2 Attack patterns and products

Figure 3.7 shows both CAPECs according to product frequencies and products clustered according to CAPEC frequencies. The clustering of CAPECs has a clear series of low-cost merges followed by a hierarchy of high cost merges. These low cost merges are accounted for since CAPECs co-occur even more frequently than products. The product hierarchy is similar to hierarchies by exploit type but there's more separation between similar products

and product types are also more split (web browsers, for example, are now divided between all three clusters).

## 3.5 Method Comparison

Figure 3.8 shows how these methods cluster the top 40 products. Some products, for example the Windows family and Android, are clustered together in all four clusterings. Other product pairs, such as web browsers Chrome, Safari, and Internet Explorer, are clustered together in three of the four clusterings. Hierarchical clustering by exploit type, in particular, is dissimilar from the other methods in that one cluster is significantly smaller than the other two, containing only 4 out of 40 datapoints.

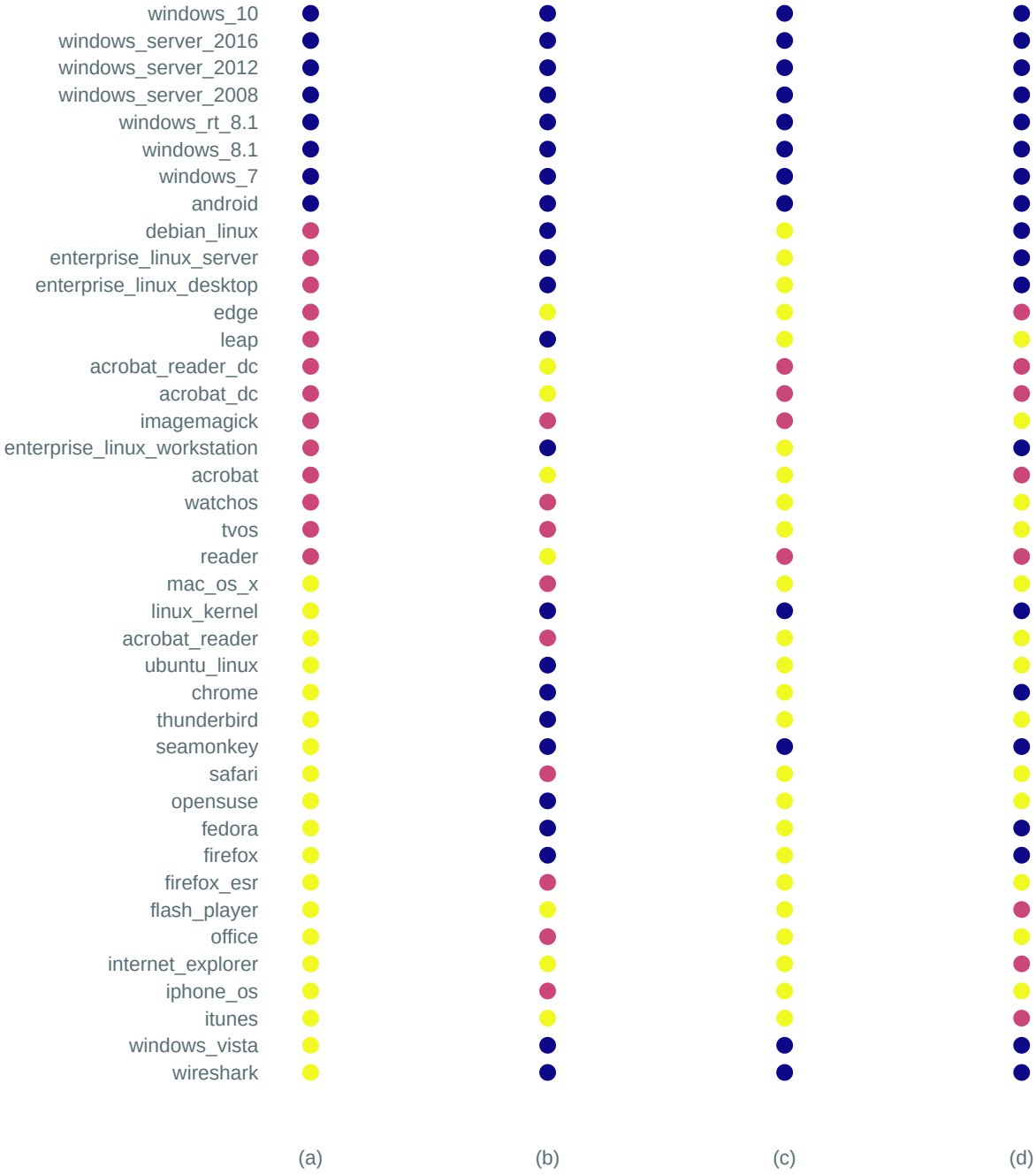



**Figure 3.8:** Cluster comparison: these are the assigned cluster labels (aligned for maximum matching) between (a) $K$-medoids by exploit type, (b) $K$-medoids by attack pattern, (c) hierarchical clustering by exploit type, and (d) hierarchical clustering by attack pattern.

# 4 Time series segmentation

Segmentation is a more specific form of the clustering problem considered in Chapter 3 [30]. Instead of grouping points in higher dimensional spaces, segmentation works to split values indexed in a one-dimensional space into groups called segments. In time series segmentation, the index is chronologically ordered.

Segmentation, the process of partitioning the data, is closely related to histograms, the resulting data representation [13]. A histogram represents data by aggregating the entries within each segment. This can be used to represent summary statistics on data streams without needing to store all of the data in memory.

Another related concept is change point detection [1], where the breakpoints represent changes in the underlying data or the process that generated it.

In this chapter, we outline these concepts and compare several algorithms for segmenting different features. Namely, we consider a dynamic programming algorithm that finds optimal segmentations for a given additive cost function, an algorithm that identifies candidate change points in exponentially distributed data, and an approximation algorithm that identifies candidate change points in multinomially distributed data.

Each algorithm corresponds with an experiment. The first experiment, segments the average severity by product using the dynamic programming solution. The second experiment frames delay as an exponential process and identifies candidate change points in the delay traces of specific products and versions. The third experiment models categorical features as a multinomial process and identifies candidate change points where the distribution of that feature changed. In this experiment, we consider exploit type and access complexity.

Together, these algorithms identify both individual points in time and patterns within products where periods of major change happen. These periods might represent a shift in attacker strategy or the life cycle of product and how vulnerabilities are identified as the product ages. In either case, this information helps to conceptualize the development of threats over the course of the history of the CVE list.

## 4.1 Segmentation

Segmentation divides the index set $I = 1 \dots N$ of a dataset $\mathcal{D} = \{x_1, x_2, \dots, x_N\}$ into $K$ segments. These segments are a set of half-open intervals $H = \{[b_1, b_2), \dots, [b_K, b_{K+1}]\}$ that cover $I$ and are disjoint, that is,

$$\Big(I = \bigcup_{h_i \in H} h_i\Big) \wedge \Big(\varnothing = h_b \cap h_a, \forall\ h_a, h_b \in H \times H : h_a \neq h_b\Big),$$

with breakpoints (boundaries) $B = \{b_1 = 1, b_2, \dots, b_K, b_{K+1} = N\}$.

Change point detection is an alternate framing of the segmentation problem. A change point is an abrupt change in the data stream.

Change point detection algorithms can be offline or online. Offline algorithms examine the entire dataset at once, whereas online algorithms only require that data entries be ephemerally stored in memory. Online algorithms are necessitated in situations where the entire data set cannot be readily stored.

The change point detection method, Multinomial Change Detection Method [25], considered in this thesis is unsupervised and part of a larger family of change point detection methods including CUSUM that consider a probability ratio. Other change detection methods instead model the data generatively under a stochastic process including methods such as Bayesian Online CPD. Other kernelized similarity methods including kcpa employ a kernel function to perform similarity based clustering. For a survey of change-point detection methods, see Aminikhanghahi and Cook [1].

## 4.2 Cost function

Typically, segmentation algorithms optimize some additive cost function, $c(a, b)$ where $a, b \in I$ and $a < b$. The cost of the segmentation $c(H)$ can then be defined,

$$c(H) = \sum_{i=1}^{K} c(b_i, b_{i+1}).$$

The optimal solution is the set of breakpoints $B$ that minimize $c(H)$. For example, to optimize the total mean squared error, we have the cost $c(a, b) = \sum_{i \in [a,b)} (x_i - \bar{x})^2$, where $\bar{x}$ is the average of $x_a, \dots, x_{b-1}$.

## 4.3 Dynamic programming solution

Identifying an optimal $K$-segmentation for $\mathcal{D}$ can be accomplished in $O(N^2K)$ time [13] using dynamic programming if the cost can be precomputed for all subintervals in $O(N^2)$ time. Let $\text{COST}(i,k)$ be the minimum cost of segmenting the closed interval $[1,i]$ into $k$ segments. Also let $\text{COST}(i,1) = c(1,i)\ \forall\ i \in I$. The recurrence that defines the algorithm is

$$\text{COST}(i,k) = \min_{1 \le j \le i} \text{COST}(j,k-1) + c(j+1,i).$$

The algorithm must iterate over all pairs of indices $K$ times in order to compute $\text{COST}(N,K)$, hence the $O(N^2K)$ overall time-complexity if the cost of a subinterval can be computed in constant time [12].

## 4.4 Monotonic Segmentation

Consider now, a feature that can modeled under an exponential distribution $F \sim \text{expon}(\lambda)$ with the probability density function:

$$f(x;\lambda) = \lambda e^{-\lambda x},$$

defined for all positive values of $x$. Our data can then be segmented using the negative log-likelihood as the cost function, $c$ (that is, we are maximizing the likelihood by minimizing the negative log-likelihood) parameterized by the maximum likelihood estimate for the scale parameter, $\lambda = \bar{x}^{-1}$ where $\bar{x}$ is the average value in the segment. To segment this feature, we find the $K$ breakpoints that minimize the cost function,

$$\sum_{i=1}^{K} \mathcal{L}(X_i; \frac{1}{\bar{x}}),$$

where $\mathcal{L}(x;\lambda)$ is the negative log-likelihood of the exponential distribution

$$\mathcal{L}(x;\lambda) = -\sum_{i=1}^{N} \log(f(x;\lambda))$$

and $X_i$ are the values indexed by $h_i$.

Monotonic segmentation is a variant of the segmentation problem where the average of each bin in the segmentation changes monotonically. This restricted problem can be solved by reducing the number of candidates only considering border indices [9, 29]. A border

index is an index where the average values of all intervals ending at the index prior to the border index are smaller than the average values of all intervals starting at the border index. Since placing a boundary at other indices would violate the monotonic constraint, the segmentation can be performed only considering border points.

Determining whether a given point is a border point can be done in $O(|\mathcal{B}|)$ time where $\mathcal{B}$ is the set of border points identified thus far. For each new observation, affix the observation to the end of the last bin. If the average of the bin now exceeds the average of the bin that precedes it, merge the two bins. Continue merging until the condition holds.

The entire sequence can then be segmented using a modification of the algorithm in Section 4.3 where only candidates are considered. This reduces the time complexity to $O(|\mathcal{B}|^2 K)$. This method is well suited to unimodal data and allows for $K$-segmentation as demonstrated by Haiminen and Gionis [9]. Though they consider a different cost function, the same algorithm applies [29].

## 4.5 Multinomial Change Detection Method

Many of the features of CVE data entries are categorical. These categorical features can be modelled by a multinomial distribution $F \sim \text{Multi}(p)$ where $p$ is a vector of the probabilities of the $S$ labels in the label set $L = \{\ell_1, \ldots, \ell_S\}$. The maximum likelihood estimate for $p$, $\hat{p}_i$ is simply the relative frequencies of the data that have appeared after $i$ observations,

$$\hat{p}_i = \left\langle \frac{\sum_{j=1}^{i} \mathbb{I}(x_j = \ell_k)}{i} : \ell_k \in L \right\rangle. \tag{4.1}$$

Multinomial Change Detection Method (MCDM) is an online change point detection algorithm that monitors the divergence between the static estimator $\hat{p}_i$ and an adaptive estimator $\tilde{p}_i$ over a data stream [25]. The adaptive estimator leverages the concept of temporally aware likelihood. That is, past examples are weighted according to a forgetting factor $\lambda_i$ and an adaptive count $n_i$ that represents the count of the examples included in the adaptive estimate. Because of this, the influence of a given example on the adaptive estimator decreases as time elapses.

The adaptive estimator $\tilde{p}_i$ along with the adaptive count $n_i$ and the adaptive forgetting

factor $\lambda_i$ are updated after each observation according to the following updates:

$$\begin{aligned}
\tilde{p}_i &= \left(1 - \frac{1}{n_i}\right)\tilde{p}_{i-1} + \frac{1}{n_{i-1}}\text{OneHot}(x_i) \\
n_i &= \lambda_{i-1} n_{i-1} + 1 \\
\lambda_i &= \lambda_{i-1} + \eta\left(\text{OneHot}(x_i)\frac{\nabla\tilde{p}_{i-1}}{\tilde{p}_{i-1}}\right).
\end{aligned}$$

where $\text{OneHot}(x_i)$ is a one-hot encoding of the categorical observation $x_i$,

$$\text{OneHot}(x_i) = \langle 1 \text{ if } x_i = \ell_j \text{ otherwise, } 0 \;\forall\; \ell_j \in L\rangle.$$

The selection of forgetting factors is explored by Bodenham and Adams [4]. The initial forgetting factor $\lambda_0$ is a ones-vector of length $S$ and the initial adaptive estimate $\tilde{p}_0$ is a zeroes-vector of length $S$. The initial list of breakpoints $B$ is empty and the adaptive count $n_0$ is initialized to a constant $S$ length vector of $\epsilon$ to avoid zero division. In our implementation, $\epsilon = \eta$, a parameter given by the user.

These adaptive variables $\lambda_i$ (and indirectly $n_i$ and $\tilde{p}_i$) are updated using gradient ascent. The gradients for $\tilde{p}_i$ and $n_i$ are also defined by a series of recursive updates:

$$\begin{aligned}
\nabla\tilde{p}_i &= \left(1 - \frac{1}{n_i}\right)\nabla\tilde{p}_{i-1} - \frac{\nabla n_i}{n_i^2}(\text{OneHot}(x_i) - \tilde{p}_{i-1}) \\
\nabla n_i &= \lambda_{i-1}\nabla n_{i-1} + n_{i-1}.
\end{aligned}$$

All gradients are initialized to zero vectors of length $S$.

Divergence between $\hat{p}$ and $\tilde{p}$ is measured using KL-Divergence introduced by Kullback and Leibler [15]. Let the KL-Divergence between $\hat{p}$ and $\tilde{p}$ after observation $i$ be known as $\kappa_i$ where

$$\kappa_i = \sum_{j=1}^{S} \hat{p}_i^{(j)} \log \frac{\hat{p}_i^{(j)}}{\tilde{p}_i^{(j)}};$$

here $\hat{p}_i^{(j)}$ is the $j$-th component of $\hat{p}_i$. When $\kappa_i$ exceeds the dynamic threshold $\epsilon_i$ where

$$\epsilon_i = \beta\left(S \max_{j \in 1\ldots S}\left(\left[\frac{\tilde{p}_i^{(j)}}{\sqrt{\hat{p}_i^{(j)}}}\right]^2\right)\right),$$

a breakpoint is added to $B$ and the estimates are reset to their initializations. No new breakpoints can be added to the list until a fixed grace period of $G$ observations has passed.

Parameters $G$, $\eta$, and $\beta$ can all be considered hyperparameters for the model. Parameters $G$ and $\beta$ both tune how frequently change points are identified. When set too coarsely,

change points can be missed (false negatives) and when set too finely, too many breakpoints will be identified (false positives).

The learning rate, $\eta$, controls how quickly the adaptive estimates react to changes in the data stream.

## 4.6 Segmenting severity aggregated by product

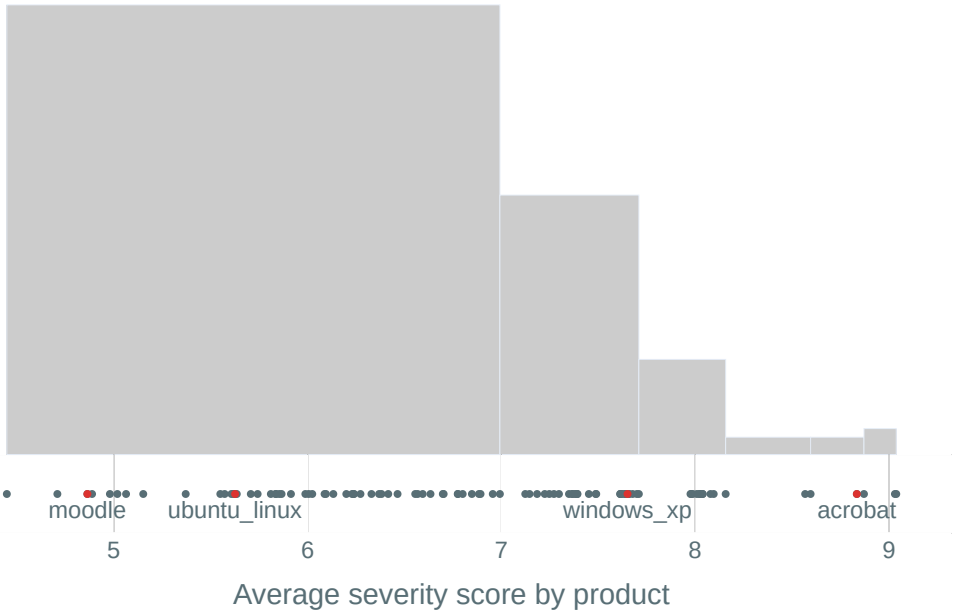



**Figure 4.1:** Severity by product: each dot on the plot represents one of the top 100 most affected products placed according to their average severity (cvss base score). Bin height represents the number of products in each bin.

Severity, as outlined in Section 2.1.5, is a quantitative summary measure of an exploit's capability and impact. In this experiment, we consider the top 100 most affected products. These products are each affected by over 200 CVE entries (Quicktime is the minimum with 201 associated CVEs).

Here we take the average severity score from all entries that affect a given product. The resulting averages can be segmented (using the algorithm from Section 4.3) to separate products into categories based on average severity.

Figure 4.1 shows a 7-segmentation of the top 100 products optimized for minimum mean squared error.

Products such as Moodle (4.86) and Ubuntu (5.62), along with about half of top products, have an average severity score in the Medium (4.0–6.9) range, and coincidentally, fall in the left most bin. Products such as Windows XP (7.65) and Adobe Acrobat (8.83) have average severities in the High (7.0–8.9) range. Adobe Air is the only product in the top 100 with an average score in the Severe (9.0–10.0) range.

The products in the three rightmost bins have a high number of exploits that allow for the remote execution of arbitrary code. For example, CVE-2014-0589 (with a cvss base score of 10.0 is a heap overflow exploit on Adobe Air and Flash Player that allowed for arbitrary code execution across a wide variety of operating system configurations. The programs in this category are mostly designed for remote management and pdf readers.

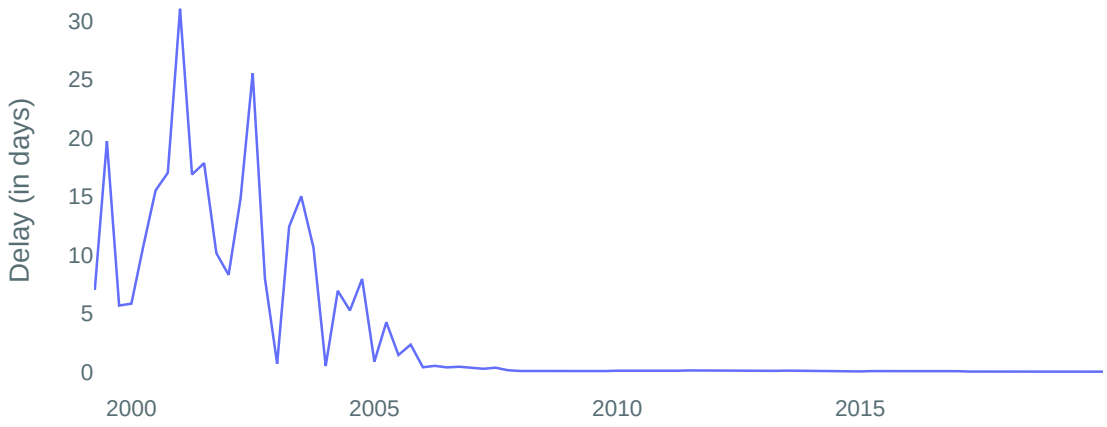
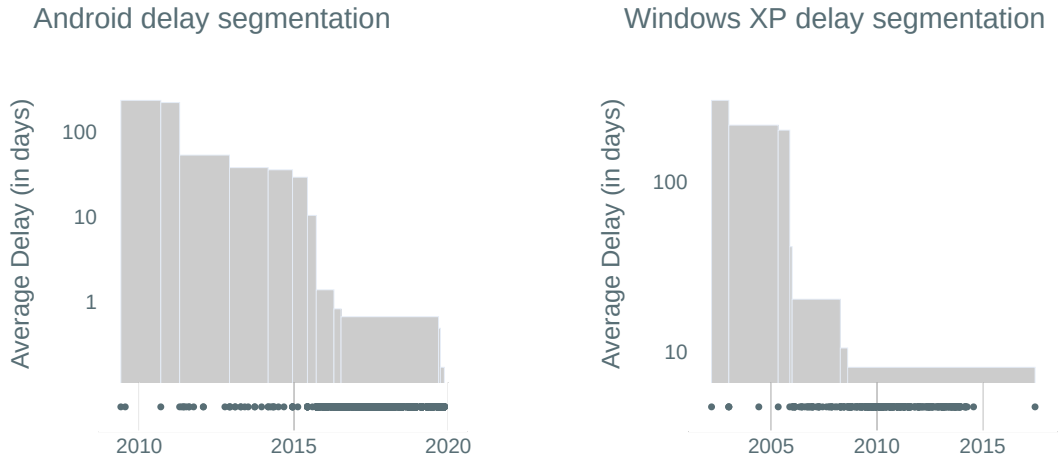



**Figure 4.2:** Delay over time: similar to trend in Figure 2.2, the delay in between entries has decreased over time. Here, delay is represented by the average delay for all entries logged per quarter.

## 4.7 Segmenting delay signals by product

Delay is the time elapsed between two observations. In CVE data, this is the time between a given CVE entry and the subsequent entry. Delay can be defined for all entries except for the most recent (since no subsequent entry is defined). The delay signal taken over the




(a) (b)

**Figure 4.3:** Product delay: the segmentations for the product delay for Android (a) and Windows XP (b). Below each histogram is a plot of the individual threats in time to visualize the distribution of publication date.

entire dataset can be seen in Figure 4.2. This experiment also considers delay traces. Here it is observed that delay can be modelled by an exponential distribution. More specifically, delay is considered to exponentially grow or decay with time depending on the product. A product can be represented a univariate data stream of delay values. Using first the algorithm from Section 4.4 to reduce the number of candidate breakpoints and then the algorithm from Section 4.3 to segment from among those candidates we can identify and summarize periods in a products' history where the pace of threats increased notably.

We will repeat this experiment for versions within a product to examine the effect of version obsolescence on delay. Note, a single CVE entry may affect multiple versions of the same product. In the intra-version experiments, entries with wildcard version CPE strings will be removed from the data since it is unclear which versions they affect.

This experiment will consider Google's Android and Microsoft's Windows XP since they both are affected by over 1,000 unique CVEs (see Figure 2.7) and represent the largest product category, operating systems. Figure 4.3(a) shows a segmentation of Android delay and Figure 4.3(b) shows a segmentation of Windows XP delay. Both products show the exponential decay of delay over time and are thus good candidates for the Algorithm in Section 4.4.

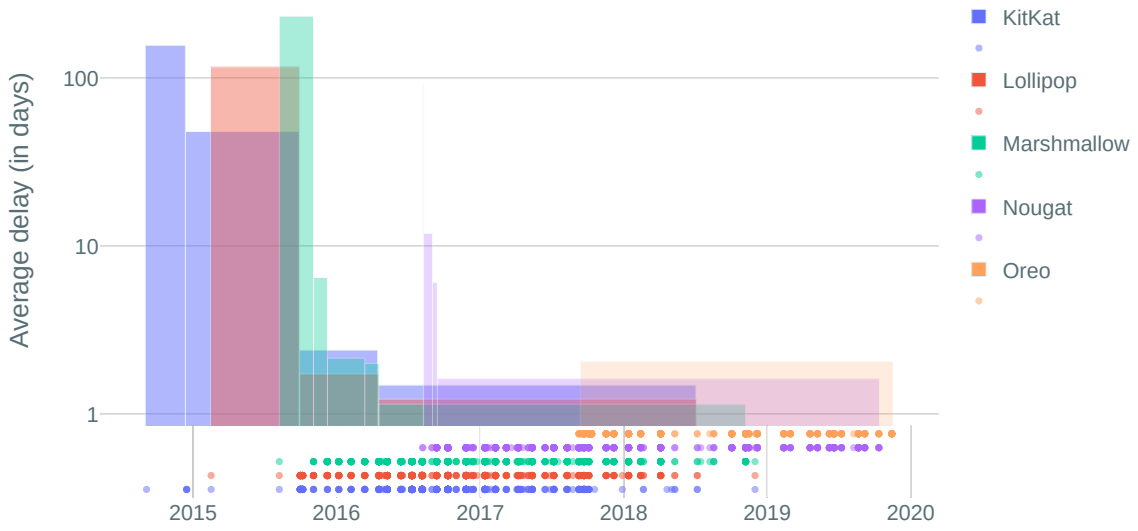



**Figure 4.4:** Version obsolescence: for version-specific CVEs, delay follows a similar pattern to the broader delay trace. Each version experiences a decrease in delay throughout its supported lifetime followed by few identified exploits after the product falls out of support.

However, its important to note that both Microsoft and Google (with an Android specific team) are CVE Numbering Authorities [19]. Reports of new threats can be added directly to the CVE list by the vendor in both cases. We also note that this trend is just as attributable as a trend in threat reporting as it is in threat proliferation.

As demonstrated in Figure 4.4, which shows the Android delay traces segmented on a version-by-version basis, threat reporting declines when a version or product becomes obsolete since the vendor is no longer providing security patches for that product. Successive Android versions also have shorter delay in initial threat reporting, which may represent a slower onset of threat proliferation or a faster identification and reporting of threats.

Overall, this experiment highlights the fact that the CVE list cannot be understood to represent all computer security threats, and is better understood as a corpus of known and researched threats since each threat has to undergo numbering, severity scoring, and exploit classification. This means that the proliferation of threats (and its slowing trend as seen in Figure 2.2) reflects both the threats themselves and the security infrastructure that has been built to detect, chronicle, and publish threat information.

Either way, the number of known threats increases with time, and for many products

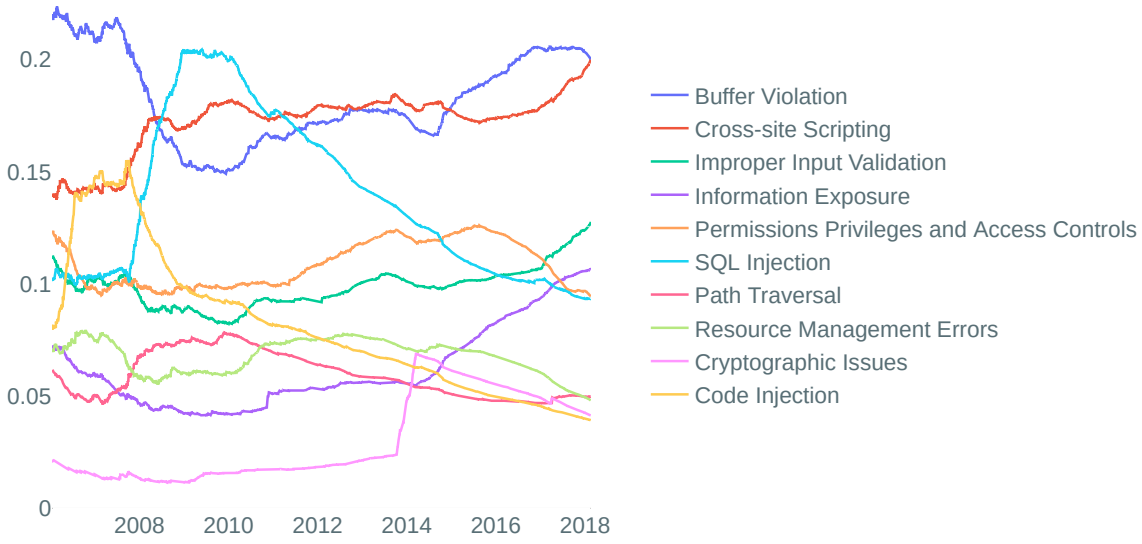



**Figure 4.5:** Static label probability estimator: The maximum likelihood estimator for $p$, $\hat{p}$ is the label's count divided by the total number of observations. Here, $\hat{p}$ is shown for the top 10 exploit classes. Entries with other exploit classes are excluded.

including both Windows XP and Android, more threats are identified as a given product ages.

## 4.8 Change point detection for categorical features

CWE codes, CAPECs, and CVSS base metrics are all categorical features. These features can be modelled as a multinomial process where each observation $x_i \sim \text{Multi}(p_i)$, where $p_i$ are the true probabilities of each label at time $i$. In this experiment, MCDM will be used to identify candidate points in time where the distribution of these features changed rapidly. These points may indicate a shift in attack paradigms or capabilities.

For CWE codes only a subset of top $S$ class labels will be used since the algorithm is designed to work with smaller label sets. To handle excluded examples, we will consider two options: removing them from the data set and grouping them under a single catch-all label.

For CWE codes, the static probability estimator $\hat{p}$ (as defined in Equation 4.1) for the representation with labels excluded is shown in Figure 4.5. This static probability trace shows the shifting relative prominence of these labels over time. Unlike Figure 2.3 however, it is a cumulative representation. Because of this, this trace is less noisy then the rolling average shown in Chapter 1 even though it has higher (observation-level) granularity.

To avoid dividing by zero in the threshold comparison, the first $j$ observations are considered a warm-up period. The index $j$ is the first index such that all $S$ labels have been observed.

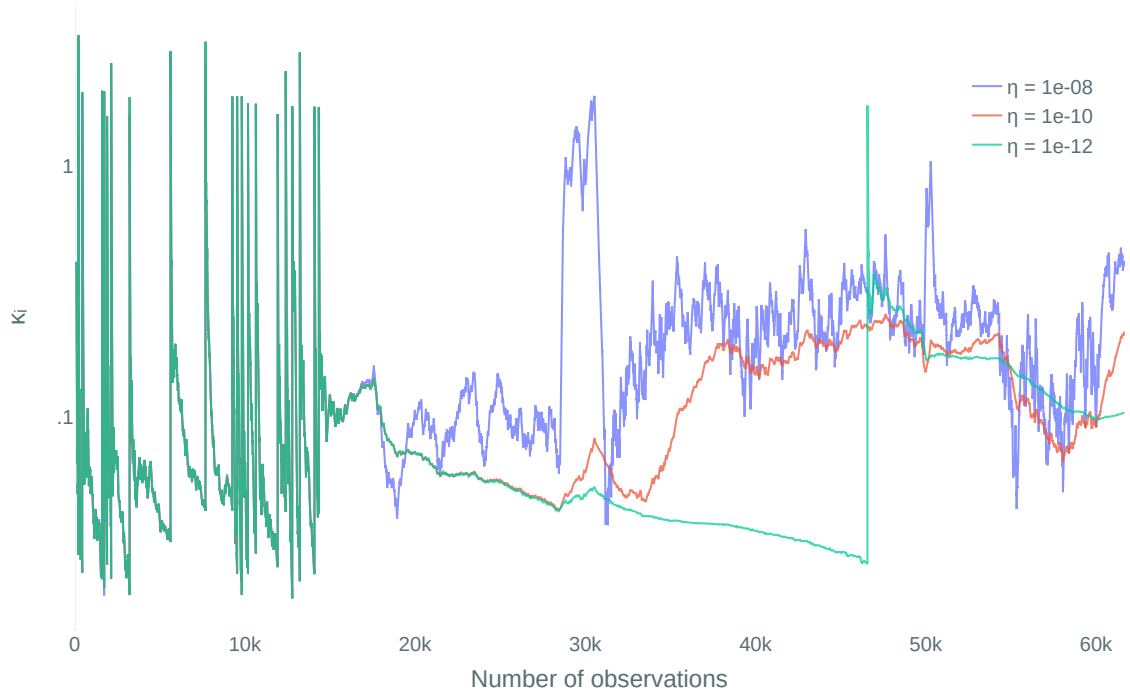



**Figure 4.6:** Learning rate: the learning rate $\eta$ affects how quickly the KL divergence $\kappa_i$ changes. Lower learning rates result in less noisy traces. Too low of a learning rate will not be able to react to changes in signal fast enough to record breakpoints.

### 4.8.1 Hyperparameter Tuning

For the third experiment, we must tune the three MCDM hyperparameters $\eta, \beta$, and $G$. Following the recommendations in Plasse and Adams [25], $\eta$ is set to $10^{-S}$ where $S$ is the number of class labels in the distribution. Figure 4.6 shows how learning rates affect the

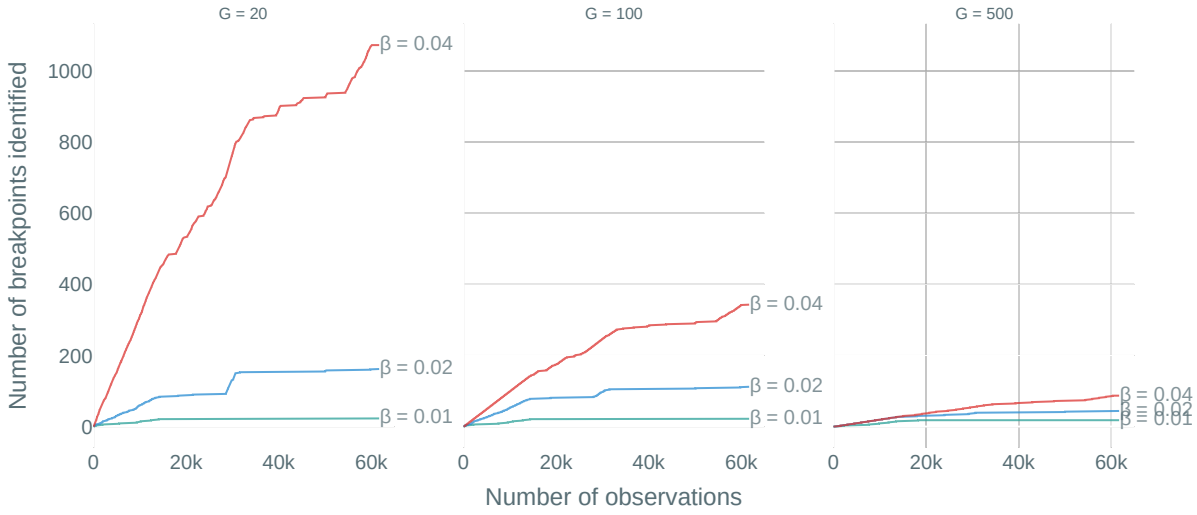



**Figure 4.7:** $\beta$ and $G$: the threshold bandwidth parameter $\beta$ and the grace period length $G$ both control how coarse the segmentation should be. Too fine a segmentation identifies false breakpoints, too coarse a segmentation neglects to identify true breakpoints. Here, $\eta$ is fixed to $10^{-10}$.

**Table 4.1:** The MCDM hyperparameters for each dataset during change point detection

| **Dataset** | $\eta$ | $\beta$ | $G$ |
|---|---|---|---|
| Top 10 CWEs (rest excluded) | $10^{-10}$ | .0216 | 700 |
| Top 10 CWEs (rest together) | $10^{-11}$ | .0216 | 1008 |
| Access Required | $10^{-3}$ | .0216 | 910 |

KL-divergence signal, $\kappa_i$. The higher learning rate results in a noisier trace which might falsely flag breakpoints by reacting too quickly to changes in the data stream. One such reaction can be seen at the 30k observation mark. The lower learning rate ceases to follow the signal since it reacts too slowly to newer observations.

The other two hyperparameters directly influence the number of change points identified and how many are false positives. More formally, Plasse and Adams [25] refers to the the average run length or $\text{ARL}_0$, the number of observations before the detector flags a false positive.

The suggested value for $\beta$ is outlined in Equation 4.2 where $\text{ARL}_d$ is the desired ARL.

$$\beta = 0.023 - 0.001 \log\left(\frac{N}{\text{ARL}_d} - 1\right) \tag{4.2}$$

For our experiments we will set our desired ARL to $n/5$ which yields a $\beta$ of .022 for the CWE experiment with $S = 10$ and other labels excluded. $G$ which also affects the number of false positives, will be set to $\text{ARL}_d/25 + w$ where $w$ is the length of the warm-up period. This way, the ARL has a strict lower bound since no breakpoints will be identified for any $i < G$. The hyperparameter values for each experiment are shown in Table 4.1

### 4.8.2 Results

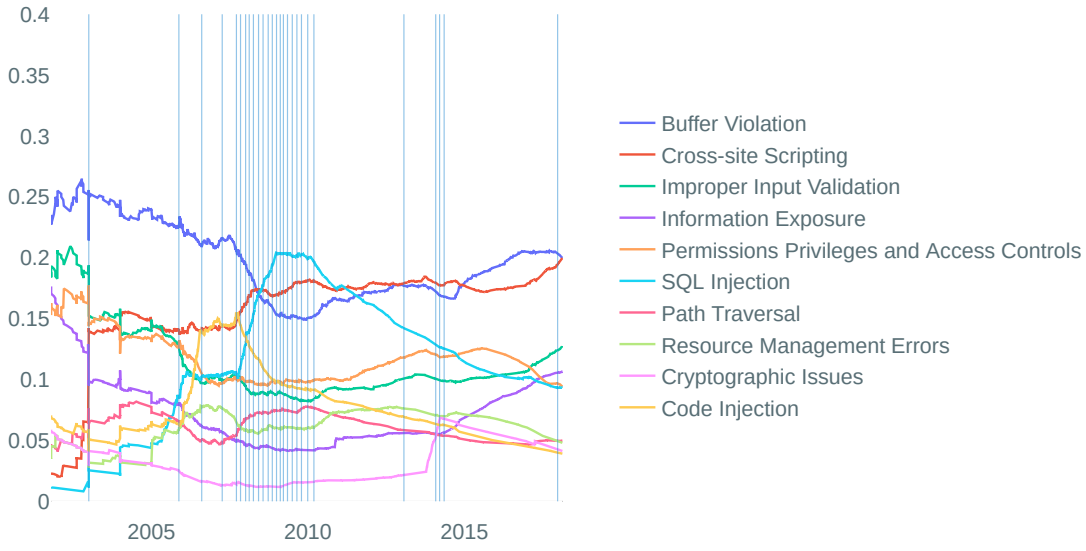



**Figure 4.8:** Top 10 CWEs: the segmented $\hat{p}$ trace from the top 10 CWEs (other labels excluded) with identified change points marked with vertical lines.

The multinomial change point detection experiments highlight periods where the composition of CVE entries changed rapidly. In Figure 4.8, we show trends among the top 10 overall CWE types. There are several periods of rapid change identified by MCDM. Two periods of sustained change, one from 2008 to 2010 and another in 2014, are marked by multiple repeatedly identified breakpoints.

The change in beginning in 2008 saw SQL injection go from representing 10% of all known threats to representing 17.5% of all known threats over a year later. At the same time, there was a more modest increase in cross-site scripting attacks and a decline in the relative prominence of buffer overflow attacks. The end of this change period marked a

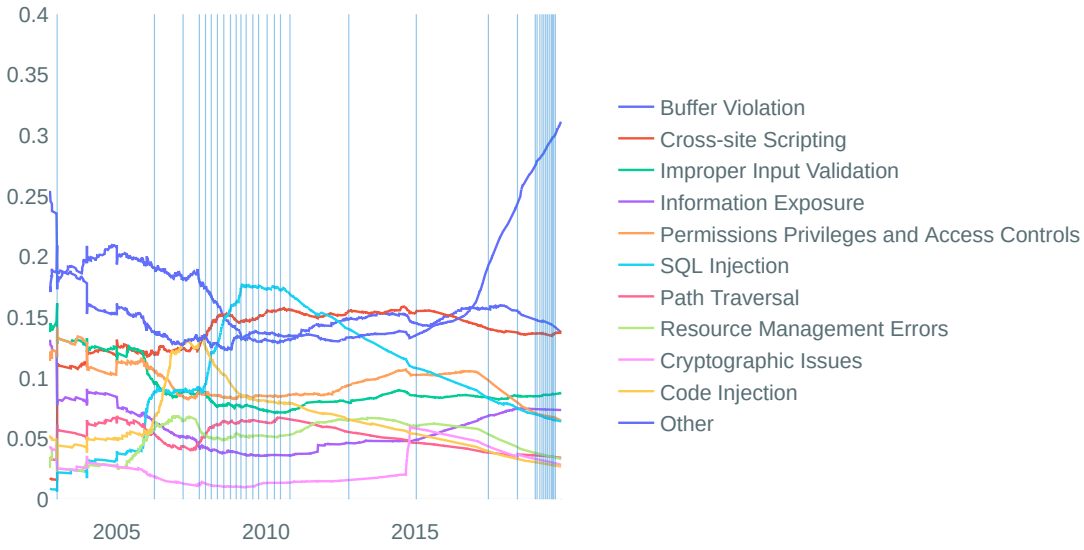



**Figure 4.9:** CWE change points: the top 10 CWEs plus a catchall category.

rapid decrease in the identification of new SQL injection attacks (shown also in Figure 2.3).

Once other attack labels are considered, we see that the relative prominence of all 10 of these known attacks are declining relative to other attacks. This change, identified by the MCDM change point detection shown in Figure 4.9 is ongoing beginning sometime in late 2016 or early 2017. During this diversification of threats, threats outside the 10 most frequent CWEs increased from 14% of all known threats in 2015 to 32% of all threats presently.

The four threats that are in the top 10 of all post-2017 CVEs that are not in the top 10 for pre-2017 CVEs are: Out-of-bounds Read (CWE-125), Cross-site Request Forgery (CWE-352), Use After Free (CWE-416), and Integer Overflow or Wraparound (CWE-190). These threats, while not new, have increased markedly in prominence over the last 3 years.

Figure 4.10 shows MCDM change point detection for the Access Complexity severity score base factor trends. This identifies a period between 2006 and 2008 as the rise of needed access in order to exploit a system. Since then, the trend has levelled, where about half of all threats require MEDIUM access and the other half require HIGH access.

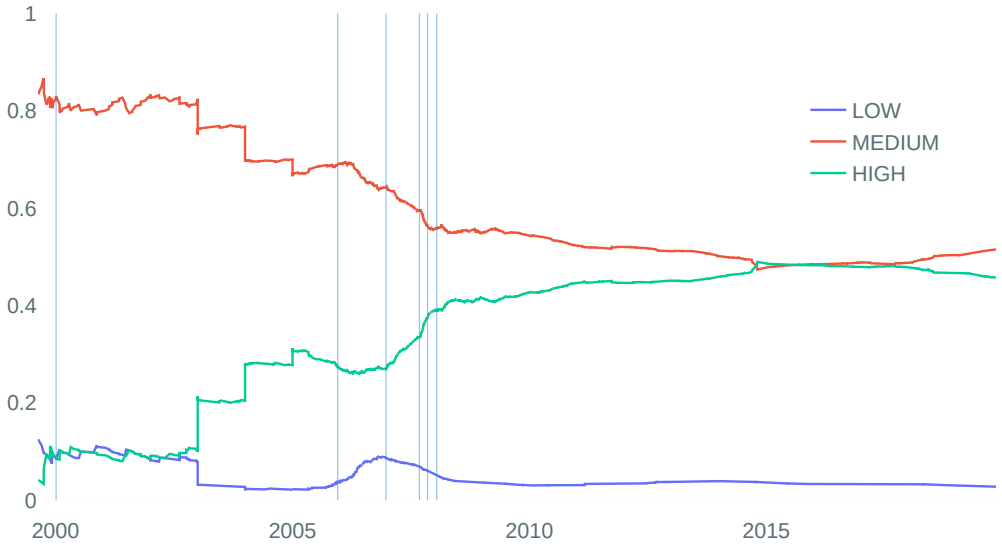



**Figure 4.10:** Access complexity: a slightly more granular version of the trace in Figure 2.5.

# 5 Further Work

There are numerous methods for approximating the problems in this thesis and some methods may be better suited to our task domain than the methods chosen here. In addition to method variety, we highlight here the data that are excluded from the analysis in this thesis.

## Clustering

PCA has a lower variance reconstruction error when more principal components are retained. This does not necessarily imply that an efficient two dimensional representation is impossible. Autoencoders, neural networks whose objective is to reproduce their input after it has been compressed into a smaller space, could be used to train a data representation with a lower variance of reconstruction error. For example, modular autoencoders (Reeve and Brown [26]), an ensemble-based autoencoder variant, or other autoencoders built for feature extraction would be more performant than KernelPCA in the two dimensional case.

Since CVE associated data (products, CAPECs, etc.) is many-to-many in nature, the CVE list can also be represented as a weighted graph (see Figure 2.9) where CVE-IDs that share associated data are connected. Alternate clustering methods, such as the Highly Connected Subgraphs (HCS) clustering algorithm [10] or CLICK, a variant of HCS that focuses on weighted graphs [28] could prove informative. Graph Laplacian matrix methods (i.e. Graph Spectral Clustering) could also be applicable.

Natural language processing techniques including topic modelling could also provide insight to how CVE entries relate to one another. In particular, automatic threat classification, as explored by Chen, Zhang, and Chen [5], could be helpful in expanding the capacity of numbering authorities.

It would also be worthwhile to use clustering techniques to further study the relationship between severity and products. Knowing which products are likely to incur severe threats could help better distribute security resources.

# Segmentation

Regarding time series segmentation, it would be useful to segment other CVE features. In particular, it would be interesting to apply the methods in this thesis to the relationship between severity and time for particular products. As products age, do they require more sophisticated attacks in order to overwhelm security countermeasures or are products late in their support life cycle just as vulnerable as older products?

Regarding methods, other approaches to segmentation include approximation algorithms, such as those summarized by Guha, Koudas, and Shim [8] which can be adapted to a likelihood-based cost function. For monotone segmentation, isotonic regression methods such as those implemented by Mair, Hornik, and Leeuw [17] can be used to identify monotonic segmentations or for candidate reduction in the same way we considered border points.

One of the broader problems in multinomial change point detection is working with larger $K$. Methods such as the algorithm developed by Wang, Zou, Yin, et al. [31] are designed to work within this constraint and would allow for the segmentation of CAPECs or even a larger set of CWEs.

It would also be interesting to synthesize different feature trace signals into a unified change detection method for the CVE overall. Methods designed to identify changes based on the homogeneity of multivariate data (see, for example, [16]) could help detect changes across the whole dataset.

# 6 Conclusion

In the 20 years since its inception, the CVE list and its associated infrastructure have been able to improve communication between computer security professionals across a variety of companies, jurisdictions, and use cases. Yet, as this shared infrastructure has grown, so has the industry's dependence on a small set of core products.

These core products each present unique security challenges, however, they occur in larger families both by function and by the vendor responsible for development. Products within these groups are affected by similar types of exploits. As demonstrated by hierarchical clustering, these groups can be identified. Lessons in securing these products can then be taken from similar products.

These products, operating systems and web browsers in particular, are ubiquitous in the lives of billions of people across the world and therefore make uniquely attractive targets for attackers to exploit. As our delay analysis shows, these products are likely to encounter an increasing number of vulnerabilities and exposures throughout their supported life cycles and the proliferation trend has continued steadily over the course of the CVE list's history.

Similarly multinomial change detection shows that composition of these threats is also changing and may even now be undergoing a paradigm shift. In particular, we identified two periods of sustained change. The first, from 2008 to 2010 saw the rise of SQL injection as the primary threat. Since 2010, SQL injection has been in a sustained relative decline. Currently, however, threats outside of the all-time top 10 now exceed 30% of all threats.

Being able to understand these trends in threat development and the relationships and correlations that underpin the co-occurrence of different exploit types is imperative to both securing products and rapidly identifying new threats. As the CVE list continues to grow, data mining tools such as the ones discussed in this thesis will prove increasingly useful for analyzing, summarizing, and visualizing threat data.

# Bibliography


[1] S. Aminikhanghahi and D. J. Cook. "A survey of methods for time series change point detection". In: *Knowledge and information systems* 51.2 (2017), pp. 339–367.

[2] J. Baarsch and M. E. Celebi. "Investigation of internal validity measures for K-means clustering". In: *Proceedings of the international multiconference of engineers and computer scientists*. Vol. 1. sn. 2012, pp. 14–16.

[3] D. W. Baker, S. M. Christey, W. H. Hill, and D. E. Mann. "The Development of a Common Enumeration of Vulnerabilities and Exposures". In: *Recent Advances in Intrusion Detection*. Vol. 7. 1999, p. 9.

[4] D. A. Bodenham and N. M. Adams. "Continuous monitoring for changepoints in data streams using adaptive estimation". In: *Statistics and Computing* 27.5 (2017), pp. 1257–1270.

[5] Z. Chen, Y. Zhang, and Z. Chen. "A categorization framework for common computer vulnerabilities and exposures". In: *The Computer Journal* 53.5 (2010), pp. 551–580.

[6] FIRST. A Complete Guide to the Common Vulnerability Scoring System Version 2.0. https://www.first.org/cvss/v2/guide [Accessed: 2.4.2020]. 2007.

[7] FIRST. Common Vulnerability Scoring System version 3.1: Specification Document. https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf [Accessed: 2.4.2020]. 2019.

[8] S. Guha, N. Koudas, and K. Shim. "Approximation and streaming algorithms for histogram construction problems". In: *ACM Transactions on Database Systems (TODS)* 31.1 (2006), pp. 396–438.

[9] N. Haiminen and A. Gionis. "Unimodal segmentation of sequences". In: *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE. 2004, pp. 106–113.

[10] E. Hartuv and R. Shamir. "A clustering algorithm based on graph connectivity". In: *Information processing letters* 76.4-6 (2000), pp. 175–181.

[11] T. Hofmann, B. Schölkopf, and A. J. Smola. "Kernel methods in machine learning". In: *The annals of statistics* (2008), pp. 1171–1220.

[12] Y. E. Ioannidis and V. Poosala. “Balancing Histogram Optimality and Practicality for Query Result Size Estimation”. In: *SIGMOD Rec.* 24.2 (May 1995), pp. 233–244. ISSN: 0163-5808. DOI: 10.1145/568271.223841. URL: https://doi.org/10.1145/568271.223841.

[13] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. “Optimal Histograms with Quality Guarantees”. In: VLDB ’98 (1998), pp. 275–286.

[14] S. C. Johnson. “Hierarchical clustering schemes”. In: *Psychometrika* 32.3 (1967), pp. 241–254.

[15] S. Kullback and R. A. Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[16] A. Lung-Yut-Fong, C. Lévy-Leduc, and O. Cappé. “Homogeneity and change-point detection tests for multivariate data using rank statistics”. In: *arXiv preprint arXiv:1107.1971* (2011).

[17] P. Mair, K. Hornik, and J. de Leeuw. “Isotone optimization in R: pool-adjacent-violators algorithm (PAVA) and active set methods”. In: *Journal of statistical software* 32.5 (2009), pp. 1–24.

[18] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. “Kernel PCA and de-noising in feature spaces”. In: *Advances in neural information processing systems*. 1999, pp. 536–542.

[19] MITRE. CVE - CNA Participants. https://cve.mitre.org/cve/request_id.html [Accessed: 4.17.2020].

[20] MITRE. CVE - Terminology. https://cve.mitre.org/about/terminology.html [Accessed: 4.3.2020]. 2017.

[21] MITRE. CVE ID Syntax Change (Archived). https://cve.mitre.org/cve/identifiers/syntaxchange.html [Accessed: 4.3.2020]. 2018.

[22] MITRE. CVE-Compatible Products and Services (Archived). https://cve.mitre.org/compatible/compatible.html [Accessed: 14.4.2020]. 2017.

[23] NIST. Common Platform Enumeration: Naming Specification Version 2.3. https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7695.pdf [Accessed: 2.4.2020]. 2011.

[24] npm. Auditing package dependencies for security vulnerabilities. https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities [Accessed: 9.4.2020]. 2018.

[25] J. Plasse and N. M. Adams. "Multiple changepoint detection in categorical data streams". In: *Statistics and Computing* 29.5 (2019), pp. 1109–1125.

[26] H. Reeve and G. Brown. "Modular autoencoders for ensemble feature extraction". In: *Feature Extraction: Modern Questions and Challenges*. 2015, pp. 242–259.

[27] E. Schubert and P. J. Rousseeuw. "Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms". In: *CoRR* abs/1810.05691 (2018). arXiv: 1810.05691. URL: http://arxiv.org/abs/1810.05691.

[28] R. Sharan and R. Shamir. "CLICK: a clustering algorithm with applications to gene expression analysis". In: *Proc Int Conf Intell Syst Mol Biol*. Vol. 8. 307. 2000, p. 16.

[29] N. Tatti. "Fast likelihood-based change point detection". In: ().

[30] E. Terzi and P. Tsaparas. "Efficient algorithms for sequence segmentation". In: *Proceedings of the 2006 SIAM International Conference on Data Mining*. SIAM. 2006, pp. 316–327.

[31] G. Wang, C. Zou, G. Yin, et al. "Change-point detection in multinomial data with a large number of categories". In: *The Annals of Statistics* 46.5 (2018), pp. 2020–2044.

[32] S. Wold, K. Esbensen, and P. Geladi. "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52.

# Appendix A CVSS Formula

The severity base score is defined as

$$\text{BaseScore} = \text{round}(((0.6 \times \text{Impact}) + (0.4 \times \text{Exploitability}) - 1.5) \times f(\text{Impact}))$$

where Impact and Exploitability are

$$\text{Impact} = 10.41(1 - (1 - \text{CImpact})(1 - \text{IImpact})(1 - \text{AImpact}))$$
$$\text{Exploitability} = 20 \times \text{AccessVector} \times \text{AccessComplexity} \times \text{Authentication}.$$

The remaining terms are defined as follows

$$f(\text{Impact}) = \begin{cases} 0 & \text{if Impact=0,} \\ 1.176 & \text{otherwise} \end{cases}$$

$$\text{AccessVector} = \begin{cases} \text{requires local access :} & 0.395 \\ \text{adjacent network accessible :} & 0.646 \\ \text{network accessible :} & 1.0 \end{cases}$$

$$\text{AccessComplexity} = \begin{cases} \text{high :} & 0.35 \\ \text{medium :} & 0.61 \\ \text{low :} & 0.71 \end{cases}$$

$$\text{Authentication} = \begin{cases} \text{requires multiple instances of authentication :} & 0.45 \\ \text{requires single instance of authentication :} & 0.56 \\ \text{requires no authentication :} & 0.704 \end{cases}$$

$$\text{CImpact} = \begin{cases} \text{none :} & 0.0 \\ \text{partial :} & 0.275 \\ \text{complete :} & 0.660 \end{cases}$$

$$\text{IImpact} = \begin{cases} \text{none :} & 0.0 \\ \text{partial :} & 0.275 \\ \text{complete :} & 0.660 \end{cases}$$

$$\text{AImpact} = \begin{cases} \text{none :} & 0.0 \\ \text{partial :} & 0.275 \\ \text{complete :} & 0.660 \end{cases}$$